US005079767A

United States Patent [19]

Perlman

[11] Patent Number: 5,079,767

[45] Date of Patent: Jan. 7, 1992

[54] METHOD OF MULTICAST MESSAGE DISTRIBUTION

[75] Inventor: Radia Perlman, Acton, Mass.

[73] Assignee: Digital Equipment Corporation, Maynard, Mass.

[21] Appl. No.: 398,472

[22] Filed: Aug. 25, 1989

Related U.S. Application Data

[63] Continuation of Ser. No. 249,958, Sep. 27, 1988, Pat. No. 4,864,559.

[51] Int. Cl.[5] H04J 3/24

[52] U.S. Cl. 370/94.3

[58] Field of Search 370/85.9, 85.13, 94.3, 370/93, 60, 94.1, 85.14, 60.1

[56] References Cited

U.S. PATENT DOCUMENTS

4,740,954 4/1988 Cotton et al. 370/62

4,760,572 7/1988 Tomikawa 370/60

Primary Examiner—Douglas W. Olms

Assistant Examiner—Wellington Chin

Attorney, Agent, or Firm—Finnegan, Henderson, Farabow, Garrett, and Dunner

[57] ABSTRACT

A method for multicast communication wherein a multicast message is distributed to all the nodes in a multicast range. If a multicast message has a multicast range which is larger than one link in the network then the message is forwarded along a unique set of pathways through the range. The unique set of pathways is called a multicast spanning tree and is unique to all nodes which can communicate directly by virtue of a list of known nodes. The network is divided into areas each of which contains a group of directly communicating nodes. A group of nodes designated as level two nodes facilitates communication between the nodes in different areas.

6 Claims, 5 Drawing Sheets

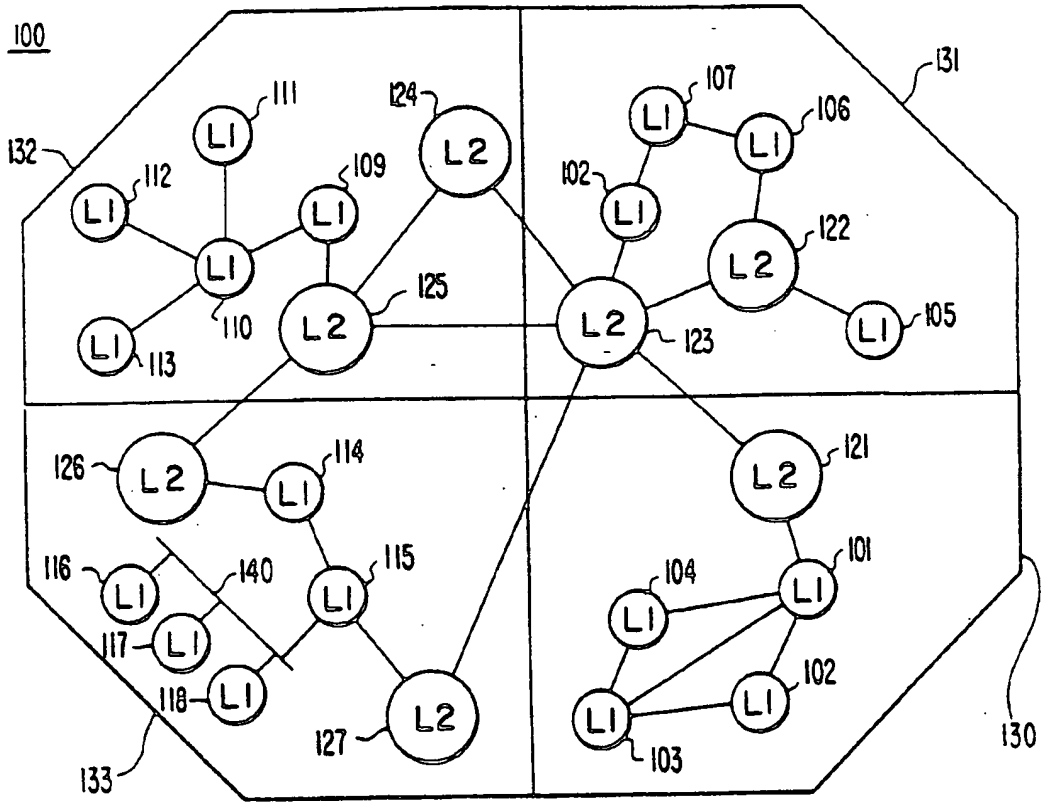


FIG. 1

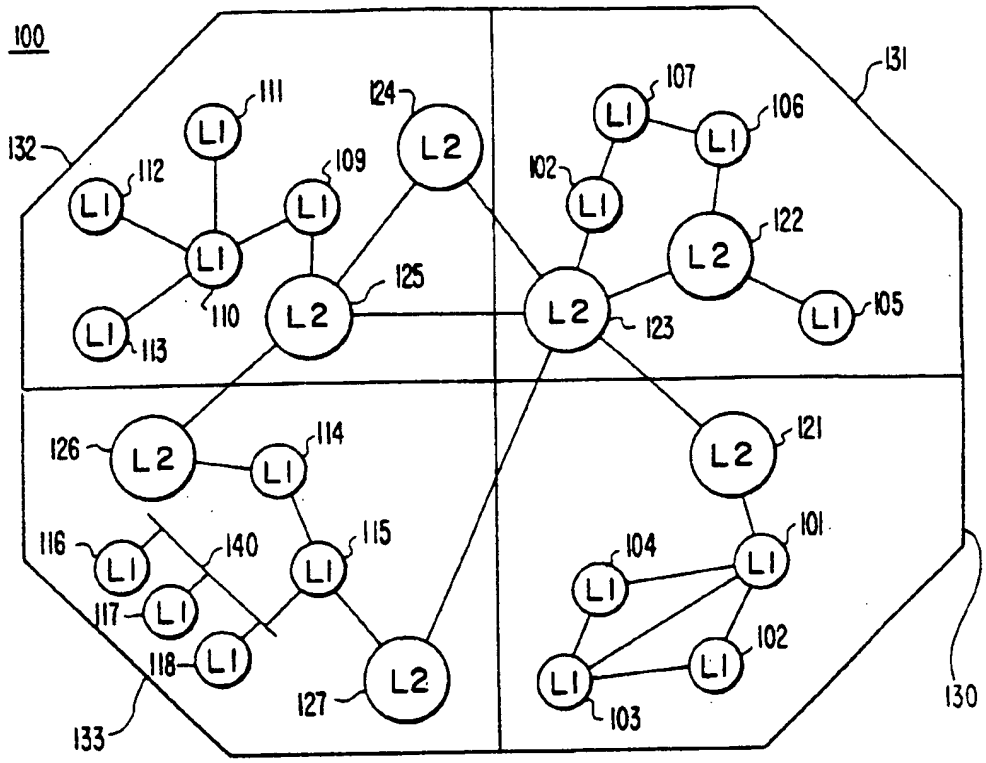

100
131
132
L1
L2
111
124
107
106
112
109
102
122
125
110
123
105
113
126
114
121
116
140
115
104
101
117
118
102
127
103
130
133

FIG. 2

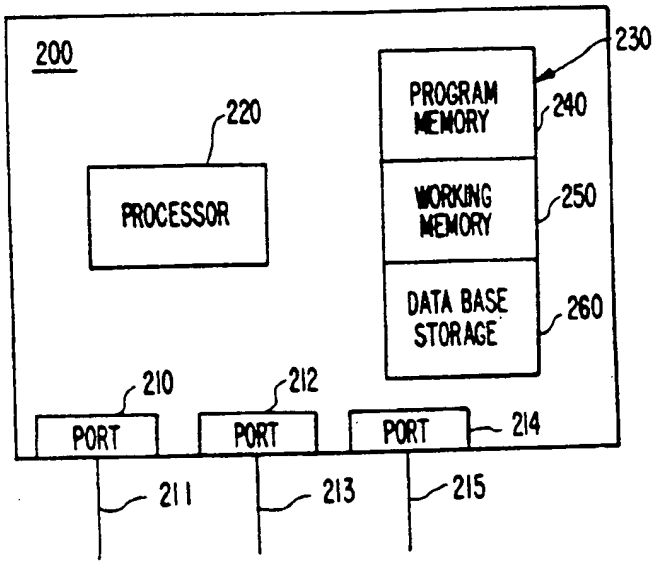

200
230
PROGRAM MEMORY
240
220
PROCESSOR
WORKING MEMORY
250
DATA BASE STORAGE
260
210
212
PORT
PORT
PORT
214
211
213
215

FIG. 3

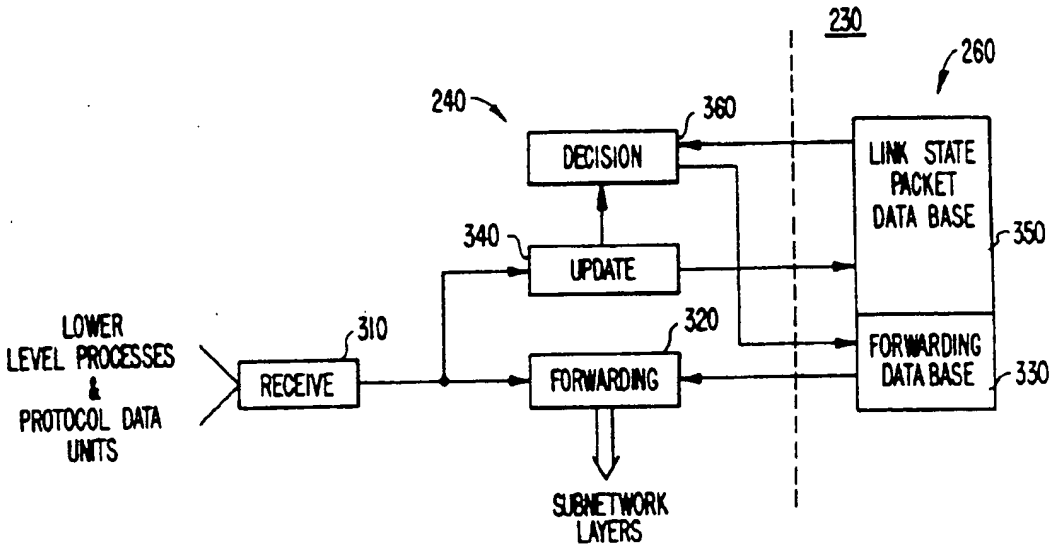

230
260
240
360
DECISION
LINK STATE PACKET DATA BASE
350
340
UPDATE
LOWER LEVEL PROCESSES & PROTOCOL DATA UNITS
310
320
RECEIVE
FORWARDING
FORWARDING DATA BASE
330
SUBNETWORK LAYERS

FIG. 4

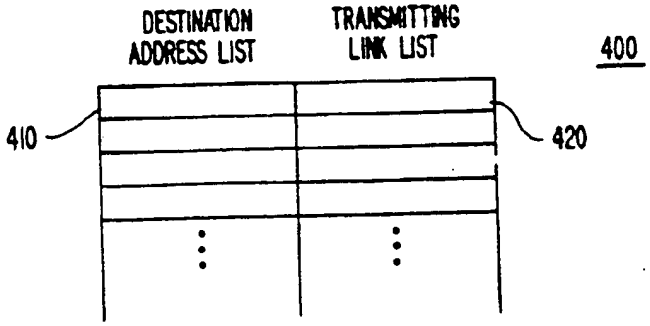

DESTINATION ADDRESS LIST
TRANSMITTING LINK LIST
400
410
420

FIG. 5

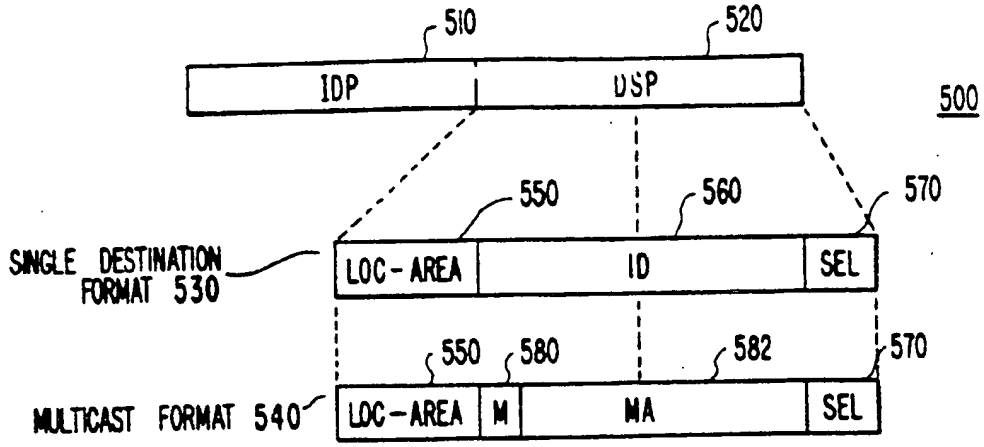


FIG. 6

MULTICAST SPANNING TREE FOR LEVEL 1 NODES

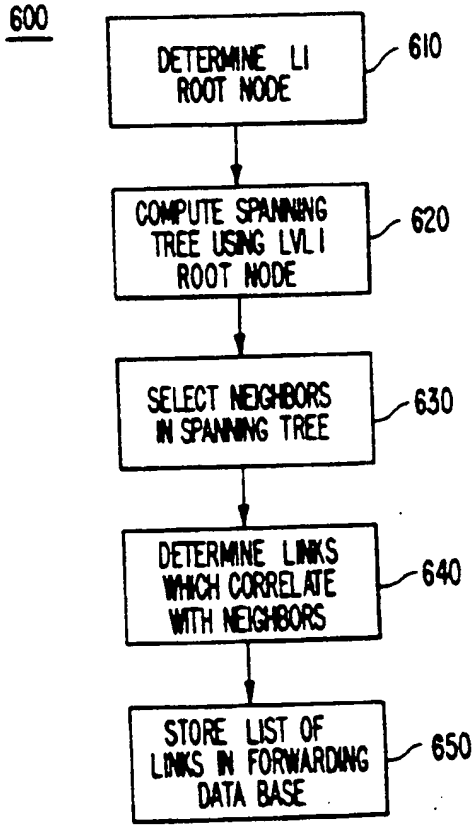


FIG. 7
LI MESSAGE RECEPTION
(LEVEL I PROCESSES)

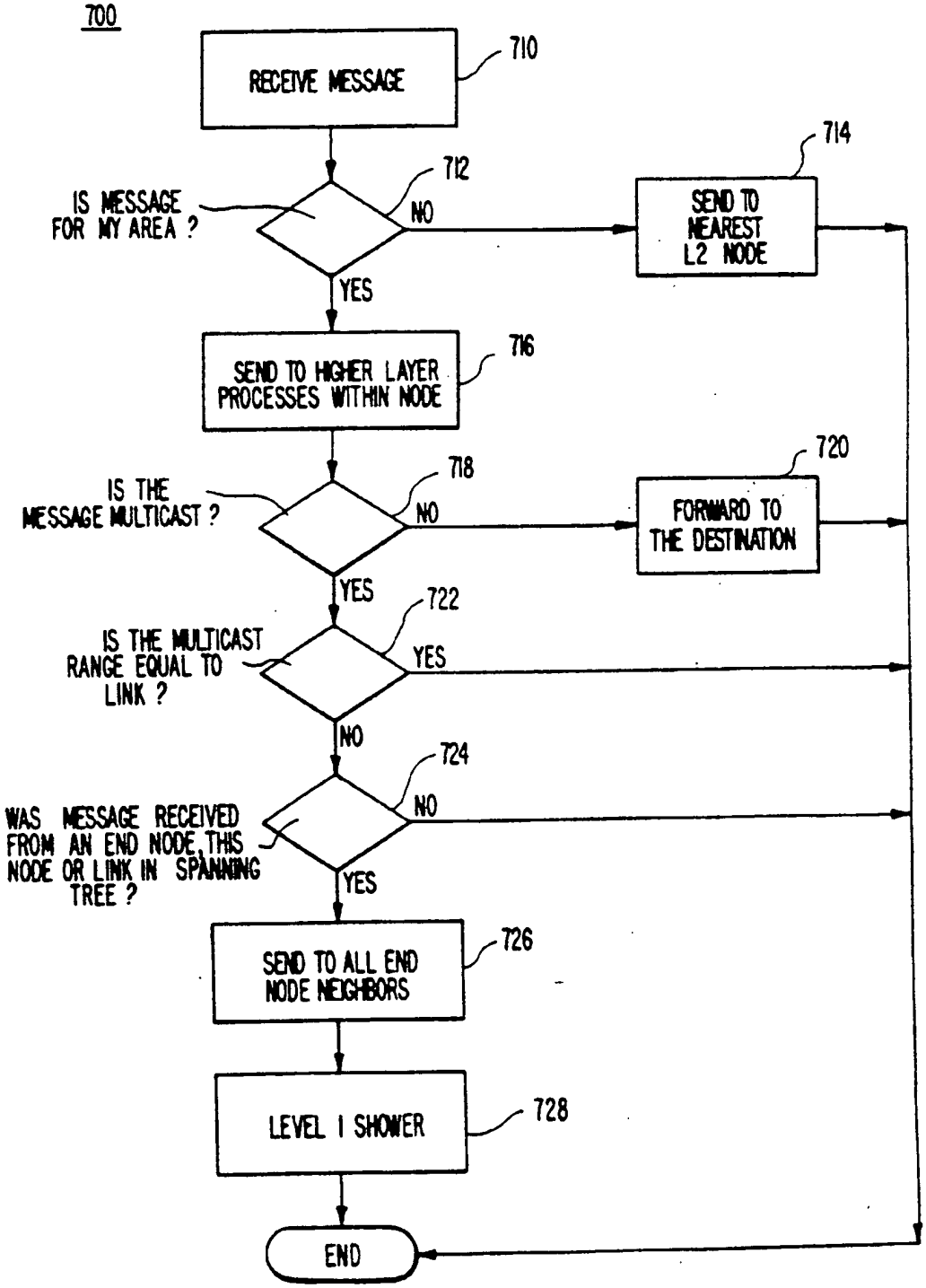


FIG. 8
SHOWER

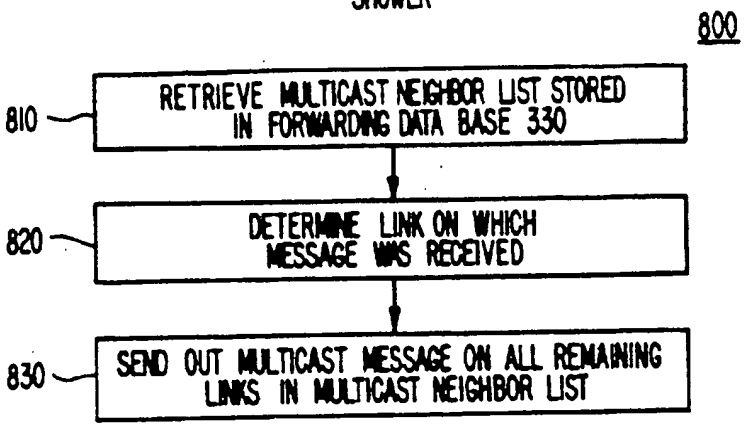
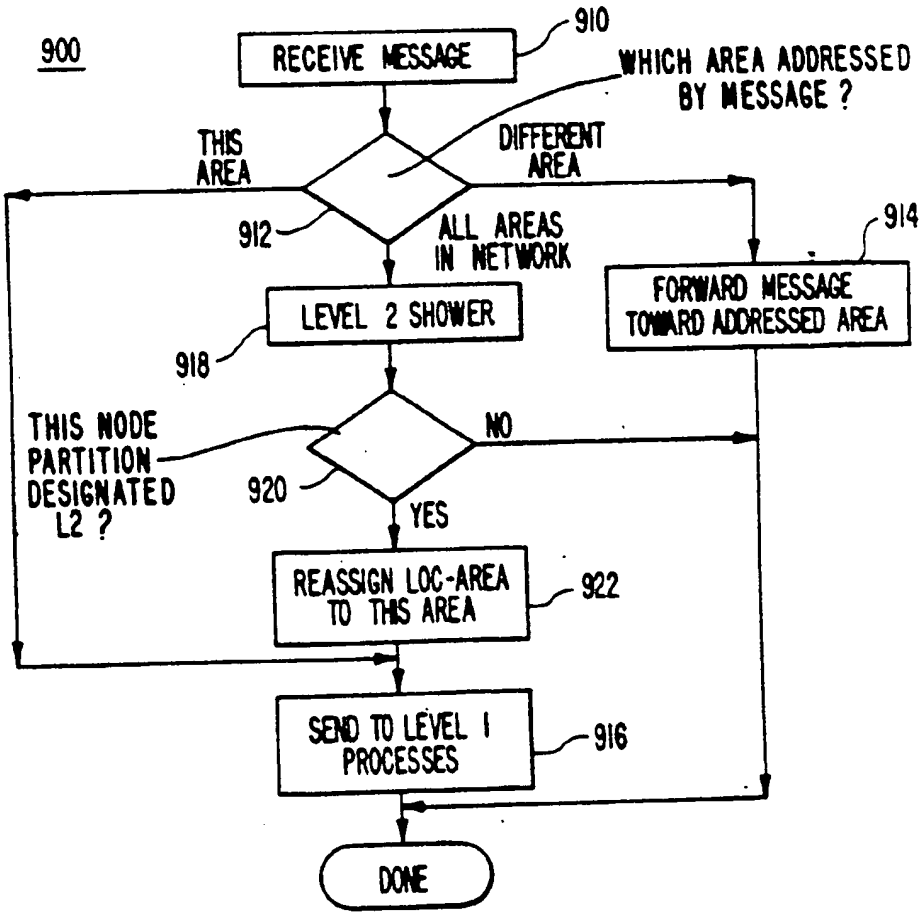


FIG. 9
L2 MESSAGE RECEPTION
(LEVEL 2 PROCESSES)



METHOD OF MULTICAST MESSAGE DISTRIBUTION

This is a continuation of application Ser. No. 07/249,958, filed Sept. 27, 1988, now U.S. Pat. No. 4,864,559.

BACKGROUND OF THE INVENTION

A. Field of the Invention

The invention relates generally to distribution of messages to a set of communication nodes within a communication network and specifically to distribution of multicast messages to a range of nodes within a network.

B. Explanation of Multicast Message Distribution

An understanding of the problems associated with transmitting multicast messages requires an understanding of networks and nodes. Hence the following explanation is given of relevant terminology and network architecture. Following that explanation is a description of other attempts at multicast message distribution.

1. Terminology

A communication network consists of "communication nodes" and "links." Generally, the term "node" is used to describe a point at which one or more functional units connect to transmission lines. For purposes of this application the term "nodes" will be used to represent the portion of a functional unit in a network which is responsible for communication on that network.

Functional units can include data processing elements designed for communication over a network such as users' data terminal equipment, computer systems running programs which require use of the transmission lines, computer systems dedicated to routing messages within the network or any combination of these possible functional units.

Communication nodes pass messages between each other using different types of communication "links." Communication links are interconnections between communication nodes, point to point connections or local area networks (LANs). Each node in a network is interconnected to at least one other node and is often connected to many nodes by links to those nodes. The nodes pass messages across links connecting the nodes.

A node can communicate "directly" with another node by knowing where that other node is and by forwarding a message to it. A set of nodes in which each node knows the location of the other nodes in the set is called a "group" of nodes. Within a group of nodes each of the nodes can forward a message to another node by forwarding the message through other nodes in the same group to a destination node not directly connected to the sending node.

If the node sends a message to another node which is not in the same group, the transmitting node can communicate "indirectly" with that other node by forwarding a message to a third node or a set of nodes which may then communicate directly with nodes in more than one group. That third node or the set of nodes in turn forwards the message to the node which is the intended destination for the message.

Usually the destination address of a message specifies a single node. In multicast message distribution a "multicast message" is meant to be received by a class of nodes. The destination address of a multicast message includes a "multicast address." A multicast address identifies a class of nodes rather than a single node. Communication nodes are usually configured to process a multicast message differently from a single destination message. Most methods of multicast distribution are inefficient and inflexible.

A multicast message is distributed to all nodes in a "range" of nodes. The multicast range is the set of all the nodes which a multicast message is to be distributed to in the network. Thus multicast message distribution enables a node to deliver a message to a whole class of nodes without knowing the members, and to find one member of a class of nodes.

The intended destination nodes for a multicast message are "logically related" nodes which define a class of nodes. That is, each of the logically related nodes is assigned a common multicast address. Nodes in the network may belong to any number of logically related classes. Therefore, a node may have any number of multicast addresses including no multicast address. A node without a multicast address does not belong to a multicast class of nodes.

When a communication node receives a multicast message, the node compares the multicast addresses of the node and the message and, if any of the addresses match, the node reads the message. Each communication node receiving a multicast message determines, according to a multicast distribution method, to which other nodes the receiving node should forward the message to ensure that all nodes within the multicast range of nodes receives the multicast message.

The range of a multicast message may not contain all of the logically related nodes which are in a class of nodes. Of course, all of the nodes in a range are not necessarily logically related.

Range selection for multicast distribution depends on which set of nodes the node sending a multicast message wants to receive the multicast message. This selection is based on economics and potential location of multicast message destination nodes.

2. Network Architecture

In order to operate a group of directly communicating nodes, each node in the group must "know" the correct directions or links to use in forwarding messages to destination nodes within the group.

In general, knowledge of which link to use in forwarding a message requires the node receiving that message to know the status of its group of directly communicating nodes. The status of those directly communicating nodes includes their current configuration or topology and an identification of which of the directly communicating nodes are currently "active" and may be used to forward a message to another node. An active node is a node capable of communicating to the other nodes in the network.

Knowledge of the status of the group of directly communicating nodes generally requires that each active node in the group know its immediate "neighbors." The neighbors of a node are those active nodes which are connected to that node by a single link.

In one mechanism for inter-node communication to provide such knowledge, each node prepares a packet of information designated as a "link state packet." The link state packet contains, inter alia, the address of the node preparing the packet, the neighbors of the node preparing the packet, and information concerning each link with that node's corresponding neighbors. All of the directly communicating nodes in a group collect these link state packets from other nodes in the group

and make a list of the nodes and their associated status information.

The link state packets are primarily used for calculation of pathways through the network. One example of such a calculation which is discussed in A. V. Aho, J. E. Hopcraft, J. D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, Reading, Mass., 1983, Dijkstra's algorithm, and which is herein incorporated by reference, is the creation of a least cost "spanning tree." The least cost spanning tree is an ordered interconnection of all directly communicating nodes which has no loops and provides the least "cost" path from the node calculating the spanning tree to every other node in a group of directly communicating nodes.

The cost of each path is determined according to criteria chosen by whomever operates the network. These criteria include such factors as the actual price of using the link which could be a leased line, or the volume of traffic through a particular link or node. For example, in order to encourage uniform utilization of the network links, the cost of high volume links would be increased.

Using link state packets, each node determines which nodes and links are active in its corresponding group of directly communicating nodes and then uses that information to determine a spanning tree. The spanning tree of each node dictates which link to use in forwarding messages to directly communicating nodes in the corresponding node group.

The directly communicating communication nodes in each group should each periodically update the status of the other directly communicating nodes by regularly creating and transmitting new link state packets and using the packets received from other directly communicating nodes to update their own spanning tree. This update process ensures, among other things, that if a node becomes inactive, a new operable path connecting every node in the group will be formed; if possible. Updating creates a problem, however. As the number of nodes in a group increases, a greater amount of time is spent calculating the necessary path information. At some point it becomes inefficient to add any more nodes to the group of directly communicating nodes.

Furthermore, greater numbers of nodes in a group require greater amounts of network resources such as processor time required for network operations and computer memory space to hold network status information.

In order to accommodate an increasing number of nodes, a hierarchical structure can be introduced into the network. For example, the network can be divided into several areas, each area can correspond to a different group of directly communicating nodes. Certain nodes will only be able to communicate directly with the other nodes in the same area and these are referred to as "level one nodes", and can be considered to constitute the first hierarchical level.

Other nodes are capable of direct communication with level one nodes in one area as well as similar nodes in other areas. These nodes are referred to as "level two nodes" and constitute a second hierarchical level. All level two nodes communicate with each other and operate in a similar manner to level one nodes communicating in the same area.

Using this scheme, level one nodes within one area may send a message to level one nodes in another area by forwarding the message to a level two node which in turn forwards the message via level two nodes to another level two node within the appropriate area. Once the message arrives in the correct area it is forwarded by level one nodes to the destination node.

3. Prior Attempts at Multicast Distribution

The multicast message must be distributed throughout a range of communication nodes which should include the corresponding nodes for which the multicast message is intended. Thus, every node in the range of nodes which receives the multicast message may be an "interested" node.

One conventional method of multicast distribution is called "reverse path forwarding." In reverse path forwarding, a node first examines a multicast message to determine whether the multicast message is received over a link which the node would normally use in forwarding messages to the node which is the source of the multicast message. If so, the node forwards that message along all of the links connected to the node except the link on which the message was received. In other words, nodes forward multicast messages received on links which correspond to the pathway in its spanning tree which includes the source of the message.

However, this method of multicast distribution is not suited for busy networks because it overloads the network with unnecessary message processing. Unnecessary message processing occurs when a message is received by a node across links not in the spanning tree. Reverse path forwarding causes unnecessary message processing because a multicast message is only accepted for forwarding when the message is from the link in the receiving node's spanning tree corresponding to the message source. This is disadvantageous because a node can receive many copies of the same message which do not require any processing but occupy resources while the node determines if a multicast message should be forwarded. High volume nodes which carry many messages during normal operation should not be burdened by unnecessary message transmissions.

This type of reverse path forwarding is also not suited to hierarchical networks since each node decides whether to forward the message based on the source of the message. In a hierarchical network, this source dependence is not feasible because, inter alia, the hierarchical levels usually are independent of each other and would not necessarily have a spanning tree containing the node.

Modifications to the reverse path forwarding approach of multicast distribution reduce unnecessary message processing but add the expense of computational time and memory space or reduce the system effectiveness. For example, an extension of the basic reverse path forwarding technique involves specifying a distance as a number of links between communication nodes which the message traverses before forwarding is discontinued. However, by limiting the number of links between communication nodes, this method produces an inflexible system which can not adapt to changes in the network topology.

All of these methods of multicast message distribution use network resources inefficiently, are unable to operate in a hierarchical network architecture, and are inflexible.

SUMMARY OF THE INVENTION

To avoid these problems and in accordance with the purpose of the invention, as embodied and broadly described here, in a communication network having a plurality of interconnected communication nodes, each

of which is directly communicating with at least one other of said nodes by virtue of a corresponding list of known nodes maintained within each node, a method of distributing a multicast message addressed to a range of nodes in said network comprises the steps of determining, by each of said nodes from its corresponding list of known nodes, at least one group of directly communicating nodes containing the node; calculating, by each of said directly communicating nodes in each of said groups, a corresponding unique set of pathways between the nodes in the corresponding group for distributing multicast messages to said directly communicating nodes in said group including the substeps of selecting one of the nodes in each group of directly communicating nodes to be a root node for that group, and calculating said unique pathways for each group to connect the root node for that group to every other directly communicating node in that group without causing said pathways to form loops; and showering said multicast message to all of said node in the corresponding range by distributing said multicast message along said unique set of pathways corresponding to the groups containing the nodes in said range.

The accompanying drawings, which are incorporated and are constituted as part of the specification, illustrate one embodiment of the invention and, together with the description, serve to explain the principles of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 shows a network configuration having nodes and links interconnecting the nodes and end nodes;

FIG. 2 is a block diagram of the elements in each of the nodes in FIG. 1 which are used to carry out a preferred embodiment of the present invention;

FIG. 3 is a block diagram of a preferred routing process carried out by nodes in accordance with this invention;

FIG. 4 is an illustration of a routing forwarding data base;

FIG. 5 is an illustration of a preferred network address format;

FIG. 6 is a flow diagram of a procedure followed by level 1 nodes in calculating the multicast spanning tree;

FIG. 7 is a flow diagram illustrating a preferred procedure for handling received messages in accordance with the method of the present invention; and

FIG. 8 is a flow diagram illustrating a procedure for implementing a shower according to the invention.

FIG. 9 is a flow diagram illustrating a preferred procedure for message reception procedures executed by level 2 processes in accordance with the present invention.

DESCRIPTION OF THE INVENTION

Reference will now be made in detail to the present preferred embodiment of the invention, an example of which is illustrated in the accompanying drawings.

A. Network and Node Organization

FIG. 1 shows a typical network configuration which can operate in accordance with the present invention. In accordance with the invention, network 100 has a plurality of communication nodes, each of which is directly communicating with at least one other of said nodes by virtue of a corresponding list of known nodes maintained within each node. Preferably, network 100 consists of communication nodes 101-118, 121-127 which are either "intermediate system" nodes 101-104, 106-110, 114-115 or "end system" nodes 105, 111, 112, 113, 116, 117, and 118. An end system node is referred to as an "end node" and an intermediate system node is referred to as a "node. An end node does not forward messages, but only generates messages and receives messages.

According to the hierarchical structure of the present invention, the network 100 is divided into areas of nodes wherein in each of the areas ones of the nodes designated as level one nodes may only communicate directly with other ones of the nodes in the same area. The areas are sometimes referred to as subdomains when the network is called a domain. Network 100 in FIG. 1 is divided into four areas: 130, 131, 132, and 133.

Each of the level 1 nodes (L1) 101-118 are in one of areas 130-133 network 100, and each level 1 node maintains a list of all other known level one nodes which are located in the respective corresponding area 130-133.

A second hierarchical level comprises second level nodes designated level 2 nodes. Level 2 nodes are distributed throughout the areas of the network and each level 2 node maintains a list of all other level 2 nodes in the network. The level two nodes constitute another group of directly communicating communication nodes. Level 2 nodes may also belong to one of the group of level 1 nodes also.

Each of the nodes in network 100 is a communication node which transmits messages across links to other communication nodes. In FIG. 1 all the links are point to point links, except link 140 which is a connecting level 1 node 115, and end nodes 116, 117, and 118.

The exchange of messages which take place in the preferred embodiment of the present invention relate to the "network layer." Network operations have several levels of protocol which are set by the International Standard Organization (ISO). The levels perform different functions such as standardizing and defining the formats used to communicate in a network. The network layer is the protocol level which governs the routing of messages through the network. Subnetwork layers control the physical transmission of the messages and other aspects of communication in a network. In the hierarchy of protocol levels the subnetwork layers are below the network layer.

According to the ISO standard, there are three different types of messages sent across the links in the network layer. Messages which carry information or data which is not related to network operation are called "data packets." Messages transferred between neighboring nodes on a regular basis which indicate that the sending node is active are called "hello" messages. Messages transmitted by every node to indicate the current status of that transmitting node and the nodes connected to the transmitting node by a link are called "link state packets." The link state packets will be described in greater detail below.

Prior to describing a method of multicast distribution in accordance with this invention, it is important to convey an understanding of the node configuration because the nodes perform the steps of the multicast method in the preferred embodiment. Although there are two different types of nodes, i.e., intermediate nodes and end nodes, for purposes of understanding the invention it is only necessary to understand the general node architecture associated with the node routing functions. A preferred embodiment of a node containing the necessary elements to carry out the method of this invention is shown in FIG. 2. Node 200, which has a unique

address in network 100, has three different ports 210, 212, and 214 connected to links, 211, 213, and 215, respectively.

Node 200 also includes a node processing unit 220, which is preferably a commercially available microprocessor but can also be a specially designed unit. Processing unit 220 may consist of one or more processors which operate solely node functions or which also execute other tasks The only necessary feature of node processing unit 220 for purposes of the present invention is that it contains sufficient circuitry and capability to perform the processes described below.

Node 200 also preferably includes a node memory unit 230 which has three portions: (1) program memory 240; (2) working memory 250; and (3) data base storage 260. Program memory 240 contains various sequences of instructions which cause the node processing unit 220 to perform necessary and desired operations. Program memory 240 can either be a ROM, PROM, or a RAM, depending upon the design considerations and expected use of program memory 240.

Node memory unit 230 also includes a working memory 250 which is preferably a RAM. The purpose of working memory 250 is to provide temporary storage, such as when executing the spanning tree algorithm or to store messages which are being forwarded through the node and received by the node.

The final portion of node memory unit 230 shown in FIG. 2 is node data base storage 260. Preferably, data base storage 260 includes a RAM containing tables or data bases, such as a routing forwarding data base. The routing forwarding data base contains a table correlating different destination node addresses to corresponding links to be used to transmit messages to those nodes. This table contains the results of the spanning tree algorithm.

An example of commercially available hardware utilized to implement node 200 the method of the present invention is the DECnet Router manufactured by Digital Equipment Corporation.

In order for the nodes to communicate with other nodes in their respective groups of directly communicating nodes, each node must be aware of the topology of its group, including the currently active nodes in the group and the current interconnection of links across which messages should be forwarded to reach a destination node. In order to determine the group of directly communicating nodes, each node must first maintain a list of active nodes which are its neighbors. The active nodes are those nodes able to communicate with other nodes in the network.

In order to determine which neighboring nodes are currently active, each node continually transmits "hello" messages along the links with which it is connected and receives "hello" messages from active nodes connected to these links. The process of determining the neighbors of a node is managed by processes in subnetwork layers which pass the information up to the network layer.

B. Node Operation

FIG. 3 illustrates the relationship among routing processes in the network layer performed by intermediate system nodes in accordance with a preferred embodiment of this invention. The processes consist of a receive process 310, a forwarding process 320, an update process 340, and a decision process 360. Preferably, all of the processes are stored in program memory 240 and are carried out by processor 220 in FIG. 2. Because end nodes only generate and receive messages and do not forward messages, they do not require the routing processes illustrated in FIG. 3. A receive process 310 receives messages generated in the subnetwork layers within its own node and network layer messages sent across the links coupled to the corresponding node.

Forwarding process 320 accepts data messages and link state packets for forwarding to the appropriate nodes across corresponding links. Link state packets are forwarded to all nodes in the network and data messages are generally forwarded to an intended destination address. Forwarding process 320 forwards non-multicast data messages by first determining whether the message is intended for its corresponding node and, if not, which link is appropriate for transmitting the message to its destination address. Forwarding process 320 accomplishes these tasks by comparing the message's destination address to destination addresses stored in a forwarding data base 330 in data base storage 260.

Update process 340 is responsible for maintaining link state packet data base 350 in storage 260, and for forwarding notification of a change in the neighbors ("neighbor event") or any of the link state packets in link state packet data base 350 to decision process 360. Link state packet data base 350 stores lists of nodes known to node 200, those nodes being the directly communicating nodes which form the group or groups containing node 200.

Update process 340 receives link state packets and neighbor events from receive process 310. A neighbor event is a notification from a subnetwork layer process that a neighbor has been added or deleted.

Update process 340 also generates the link state packet for its node. The link state packet contains a list of active neighbors of the node generating the packet. That list is referred to as the local link status. In generating local link status, update process 340 adds neighbors to and deletes neighbors from the node's link state packet in response to a neighbor event. If a "hello" message is received from a previously inactive link, a neighbor event notifies update process 340 to add the node which generated the "hello" message to the list of neighbors in the link state packet. If no "hello" message is received from a previously active node for a given period of time, a neighbor event notifies update process 340 to delete the neighbor from the node's link state packet. When the local link state changes or when update process 340 detects a variation in the link state packet data base update process 340 notifies decision process 360

Decision process 360 is responsible for calculating a spanning tree using Dijkstra's algorithm to determine the routing forwarding data base stored in forwarding data base 330. A spanning tree is an ordered interconnection of all nodes in the network. That is, all of the nodes out from a root node are arranged in order so that there are no loops within the set of pathways through the nodes.

The calculation of the spanning tree requires that one node in each group be selected as a root node from which the set of pathways can be determined. To maintain the routing forwarding data base for single destination messages, decision process 360 first calculates the spanning tree using its own node as the root.

In order to compile a routing forwarding data base, decision process 360 preferably utilizes the spanning tree to form a table such as data base 400 shown in FIG.

4, which preferably represents a portion of forwarding data base 330. Data packet forwarding data base 400 has two lists, destination address 410 and transmitting link 420. Destination address list 410 contains addresses of known nodes and is correlated with transmitting link list 420 which identifies links to correspond with the known nodes. Forwarding process 320 uses data base 400 when it receives a message from receive process 310 to determine which link from transmitting link list 420 forwarding process 320 should use to forward the message via the least cost pathway. The details of receive process 320, forwarding process 310, update process 340, and decision process 360 will also be discussed with respect to the multicast distribution method of the present invention.

C. Node Hierarchy

As the number of directly communicating communication nodes in a group increases, the time that it takes decision process 360 to implement Dijkstra's algorithm increases at a rate faster than the increase in the number of links. Moreover, the memory area necessary to store the link state packet data base also increases when more links and nodes are present. Consequently, a hierarchical structure has been introduced to the network of the preferred embodiment of the present invention.

Each of the nodes in the network, except for the end nodes, has the ability to forward messages to any other node, including end nodes, within its area. Most level 2 nodes 121-127 are also able to forward messages not only within their area but also to other level 2 nodes outside of their respective areas since level 2 nodes can belong to two groups of directly communicating nodes. Some level 2 nodes are not linked to any level 1 nodes and therefore only communicate with other level 2 nodes.

In order to have the dual capability, the level 2 nodes linked to level 1 nodes have two sets of receive process 310, forwarding process 320, update process 340, and decision process 360 corresponding to level 1 and 2 processes. Thus, some level 2 nodes must also maintain two lists of known active nodes for the respective groups.

The two sets of processes in the level 2 nodes are nearly identical. The level 1 processes occurring in a level 2 node are slightly different because in level 2, node messages which are forwarded to nodes which are not in the local area must be passed from the level 1 processes to the level 2 node processes. The level 2 processes then pass messages to the level 1 processes when messages intended for a node in the local area are received from outside of the local area.

Level 2 nodes forward messages to the appropriate areas. Because level 2 nodes constitute a group of directly communicating nodes within network 100, each level 2 node must be aware of the status of the active level 2 nodes within network 100. To obtain such awareness, level 2 nodes have a link state packet data base of level 2 link state packets, and a forwarding data base which correlates destination addresses of level 2 nodes with links across which messages should be transmitted to those level 2 nodes. The spanning tree algorithm is computed by each level 2 decision process so that a level two routing forwarding data base may be calculated

D. Message Formats

Network addresses for messages contain a number of portions which are illustrated in FIG. 5. The ISO format for a network address 500 contains an initial domain part (IDP) 510 and a domain specific part (DSP) 520. Initial domain part 510 designates a network identifier. For example, network 100 illustrated in FIG. 1 will have a specific address among a number of other possible networks as specified by the initial domain part 510.

Domain specific part (DSP) 520 is defined in a general single destination format 530 and a multicast address format 540. For the single destination format 530, DSP 520 includes a local area (LOC-AREA) identifier 550, an identifier (ID) field 560, and a select field 570. For the multicast address format 540, DSP 520 includes LOC-AREA 550, multicast bit 580, multicast address 582, and select bit 570.

In both the single destination and multicast address formats 530 and 540, respectively, each area of the network is uniquely defined by the combination of IDP 510 and LOC-AREA 550. LOC-AREA 550 uniquely defines the areas within individual networks.

Select field 570 uniquely identifies each of several modules which may be associated with an addressed node. A module can be any compatible subsystem such as a specific aspect of a system utilizing a node or a set of data terminal equipment to which a message may directed.

In single destination format 530, ID field 560 indicates the address of the node to which the message is sent. In the multicast destination format, ID field 560 includes, as the first bit, a multicast bit 580. Multicast bit 580 indicates that the data message should be distributed according to multicast distribution. ID field 560 also includes multicast address field 582. In multicast distribution, address field 582 contains the multicast address which a node uses to test whether the multicast address is of interest to the node. ID field 560, which is preferably defined according to IEEE 802 standard, multicast address 582, corresponds to the "group address".

When a node sends a multicast message, the sending node selects a corresponding multicast range which should include nodes having a corresponding multicast address. According to the present invention the multicast range may be a link (i.e., identifying all nodes connected to that link), an area (i.e., all nodes in that area), or the entire network. If the range is equal to a single link, only nodes connected to the single link will receive the multicast message and none of the nodes receiving the message will forward it along another link. A single link can be the multicast range for a LAN having a plurality of intended nodes.

Accordingly, the multicast range is specified in the address portion of respective multicast messages. In the preferred embodiment of the invention, two values for local area field 550 have been reserved for multicast range identification. For example, if local area field 550 equals zero then the multicast range is equal to a single link. If local area 550 equals all ones, then the multicast range is the entire network. All other multicast messages will be distributed to the area specified in the local area field 550.

E. Multicast Message Forwarding

According to the invention, each of the nodes in the network determines, from its corresponding list of known nodes, at least one group of directly communicating nodes containing the node. In the preferred embodiment, level 1 processes use a level 1 list of known active nodes in link state packet data base 350 and level 2 processes utilize a level 2 list of known nodes also stored in link state packet data base. Each of the nodes in a group is able to communicate directly with the

other nodes in the group by virtue of knowing which active nodes are in a direct communication group with the corresponding node.

Then, according to the present invention, each of the group of nodes calculates a corresponding unique set of pathways between the nodes in the corresponding group for distributing multicast messages to the directly communicating nodes in the group. In the preferred embodiment of the present invention, the corresponding unique set of pathways is a multicast spanning tree constructed for the appropriate group of directly communicating nodes. In calculating the multicast spanning tree, each node in a group uses a common root node and therefore calculates a single spanning tree for each corresponding respective group.

FIG. 6 illustrates a preferred procedure 600 followed by level 1 nodes in calculating the multicast spanning tree. Preferably processor 220 would execute procedure 600 stored as a program in program memory 240. Multicast spanning tree procedure 600 begins with all the level 1 nodes in an area selecting a root node for the spanning tree calculation (step 610). The root node is preferably selected using some simple criterion such as the lowest or highest address value.

Using the root node for an area, each level 1 node then calculates the multicast spanning tree for that area (step 620). The nodes preferably compute the multicast spanning tree in the same manner that they calculate the routing spanning tree for the routing forwarding data base (i.e., Dijkstra's algorithm) with one exception. For the multicast spanning tree, instead of each node using itself as the root node, the common root node is used.

Using decision process 360, level 1 nodes select their neighbors in the multicast spanning tree (step 630). The multicast neighbors of a node are those active nodes which are connected to that node by a single link along the multicast spanning tree.

In using the neighbors selected for the multicast spanning tree, the level 1 nodes then correlate the links to those neighbors in the multicast spanning tree using decision process 360 (step 640). The list of the correlated links is stored in the forwarding data base (step 650) and used to forward multicast messages as will be described below.

The level 2 nodes in a group of directly communicating nodes also calculate a multicast spanning tree for all level 2 nodes. As was discussed previously, each level 2 node contains a set of level 2 processes virtually identical to level 1 processes. The level 2 processes calculate a level 2 multicast spanning tree using the same steps as the level 1 nodes calculate a level1 spanning tree, except that a level 2 root node is selected rather than a level 1 root node.

As with the routing forwarding data base, the multicast spanning tree and associated table is intermittently updated throughout operation of the network in response to changes in the network topology. An updated table of appropriate links within the multicast spanning tree is needed for the remainder of the method according to the present invention.

According to the invention, multicast messages are showered to all of the nodes in the corresponding multicast range by distributing the multicast messages along the unique set of pathways, e.g., the multicast spanning tree corresponding to the groups containing the nodes in the range. Preferably, the showering of multicast messages to all nodes in the corresponding multicast range follows the procedures outlined in the flow diagram of FIG. 7.

FIG. 7 illustrates a procedure 700 of the method of the present invention which is followed when a message is received by a level 1 node. After receiving a message (step 710), the level 1 node in a hierarchical network must determine whether the message is for its own area (step 712). If the message is not for the corresponding area of the receiving node, the message is then sent to the nearest level 2 node (step 714), which is identified by its position in the spanning tree. For example, the nearest level 2 node may be the current node in which the level 1 processes are executing the procedure of level 1 message reception. After the message is sent to the nearest level 2 node (step 714), the level 1 reception procedure is complete (step 799).

If the message is for the node's area, that message is then sent to higher layer processes within the node (step 716). These higher layer processes in each node determine whether the message is intended for that node. The network layer is not concerned with whether the users or other equipment in the node is interested in the message, the network layer only controls routing of messages through the network. The modules which share the node determine if their normal destination address or multicast addresses are equal to the single destination address or multicast address of the multicast message received at the higher protocol layers. If the multicast message received is of interest to the modules of the node, then it is read by those modules.

If the receive process determines that the message is not a multicast message (step 718), then the node forwards the message to the addressed destination through the group of nodes in an area using the link identified in forwarding data base 330. After such forwarding, procedure 700 is complete (step 799).

If the message is multicast (step 718), and if the multicast range is equal to a single link (step 722), then no other nodes besides those connected to the link on which the multicast message was forwarded should receive the message. In that case, the receiving node is not required to forward the message, and is finished with the message reception process (step 799).

If the multicast range is larger than a single link (step 722), the node checks whether the multicast message was received from an end node, its own node or a link in the spanning tree (step 724). If the source of the message is not one of the sources for which the multicast distribution method is intended to forward, then message reception process 700 is complete (step 799). The message source could be the node itself if, for example, the node is executing the level 1 message reception within the level 1 processes. This could occur when the message was generated by internal modules, such as data terminal equipment for node users.

The node receiving the multicast message then sends the multicast message to all end node neighbors (step 726) and implements a level 1 shower (step 728). Preferably, this includes sending the message across every link connected to the corresponding node which is in the multicast spanning tree, except for a link within the spanning tree on which a message has arrived.

Procedure 700 also corresponds to a procedure which may be used in a non-hierarchical group of directly communicating nodes, i.e, a network having no level 2 nodes. When the method of the present invention is utilized in a non-hierarchical network, the proce-

dure is changed by deleting steps 712 and 714 which correspond to level 2.

FIG. 8 illustrates a procedure for implementing a shower according to a preferred embodiment of the present invention. Shower procedure 800 is implemented by both level 1 and level 2 processes. First a forwarding process such as, level 1 forwarding process 320, retrieves the level 1 multicast neighbor list stored in a forwarding data base, such as data base 330 (step 810).

The forwarding process then determines, from the receive process such as receive process 310, on which link the multicast message was originally received (step 820). The multicast message is then forwarded along all the links in a multicast neighbor list, such as the level 1 multicast neighbor list, except the link on which the message was received (step 830). The level 1 multicast neighbor list is the list of all the links correlated to the neighbors in the level 1 multicast spanning tree. Since all the nodes in a group are interconnected by a multicast spanning tree, all the nodes in the area will receive the multicast message without any redundant transmissions.

As explained above, in the preferred embodiment of the present invention, a second hierarchical level exists in which the level 2 nodes constitute a group of directly communicating nodes which may communicate with each across area boundaries. The second hierarchical level, or level 2, facilitates having a network multicast range option for distributing multicast messages to all nodes within the entire network.

Level 2 message reception procedures are illustrated by flow chart 900 in FIG. 9. These message reception procedures are implemented by level 2 processes. Accordingly, the step of calculating a unique set of pathways, i.e., the multicast spanning tree, includes calculating a level 1 spanning tree for each corresponding area and calculating a level 2 spanning tree for all level 2 nodes in the network by each level 2 node in the network. The calculation of a level 2 spanning tree in the preferred embodiment includes choosing a root node common to all level 2 nodes, and using the root node for the spanning tree calculation. Similar to the level 1 calculation of interconnections, the level 2 processes preferably use Dijkstra's algorithm as applied to the root node and create a list of the links connected to the corresponding node which are in the spanning tree.

After a message is received (step 910), the level 2 node executing level 2 processes determines the multicast range of the message (step 912). If the message is addressed to a different area, then the message is forwarded conventionally towards the addressed area using the routing spanning tree and corresponding data base (step 914). If the message is addressed to the area corresponding to the location of the receiving level 2 node, then the level 2 node simply sends the message to the level 1 processes within the level 2 node (step 916). The level 1 processes then implement message reception procedures, such as they are illustrated in FIG. 7.

If a message is addressed to all areas in the network, i.e., the message has a network range, then a level 2 shower is implemented in much the same manner as the level 1 shower is implemented (step 918). The one difference is the use of a level 2 multicast neighbor list which is the list of all the links correlated to the level 2 neighbors in the level 2 multicast spanning tree.

In order to distribute multicast messages to each node within the areas of the network, selected level 2 nodes are designated as partition designated level 2 nodes which serve as, inter alia, entry points into the level 1 multicast spanning tree. In the preferred embodiment of the present invention there is a single partition designated level 2 node selected for each group of directly communicating nodes in an area. The partition designated level 2 node for an area is selected from all level 2 nodes in that area based on the address of such nodes. For example, the level 2 node having the largest or smallest address may chosen as the partition designated level 2 node.

Each link state packet, then identifies the sender of the packet as either a level 1 or level 2 node, and indicates which level 2 node has been selected by that node as a partition designated level 2 nod . Decision process 360 for the level 2 decision process selects the partition designated level 2 node from the link state packets generated by level 2 nodes.

If a level 2 node receiving a message is a partition designated level 2 node (step 920), then the area addressed by the multicast message is assigned to be the corresponding area of the partition designated level 2 node (step 922). The assigned multicast message is passed down into the level 1 processes illustrated in FIG. 7 eventually causing a level 1 shower to be implemented.

With the minimal computation and memory to generate the multicast spanning trees and select the partition designated level 2 node, the method of the present invention uses a relatively small amount of the network resources. In addition, the inefficiencies of redundant message transmissions are avoided by using only those links in one spanning tree, the multicast spanning tree, to distribute the multicast messages along each hierarchical level.

Using the partition designated level 2 node in conjunction with the addressing techniques shown for the preferred method of multicast message distribution makes the method almost transparent to the nodes in the network. The simplicity and transparency of the method according to the present invention make it ideal for use in a hierarchical network architecture.

It will be apparent to those skilled in the art that various modifications and variations can be made in the multicast method of the present invention without departing from the scope or spirit of the invention. It is intended that the present invention cover the modifications and variations of this invention, provided they come within the scope of the appended claims and their equivalents.

What is claimed is:

1. In a communication network including a plurality of communication nodes interconnected by a plurality of communication links, a method of distributing a multicast message to each of the communication nodes, the method being executed by each of the nodes and comprising the steps of:

selecting a multicast pathway constructed from a set of said links, said set of links being selected such that said multicast pathway contains no loops, and such that each of said communication nodes is connected to at least one of said links in said set and is able to send a message along the links within the set to any other of said communication nodes, the selection of the multicast pathway including the substeps of

selecting one of said communication nodes as root node, and

selecting said set of links to form a spanning tree for said plurality of communication nodes using the selected root node;

receiving the multicast message by each of the communication nodes from a corresponding receiving one of said of links; and

forwarding the received multicast message along every link, other than the corresponding receiving link, which is within the set and which is connected to the node forwarding the message.

2. The method of claim 1 wherein the multicast message includes a range to which the multicast message is addressed, and

wherein said the forwarding step includes the substep, executed by each node receiving the multicast message, of

determining whether the node receiving the multicast message is within the range of the multicast message.

3. The method of claim 1 wherein said network is divided into areas of communication nodes such that in each of said areas ones of nodes designated as level one nodes may only communicate directly with other ones of said nodes in the same area, and ones of said nodes designated as level two nodes may communicate directly both with level one nodes within the same areas and with other ones of said level two nodes throughout the network; and

wherein said forwarding step further includes the substeps of

forwarding, by each of said level one communication nodes, said multicast message to one of said level two communication nodes in the corresponding area if said multicast message is not addressed to any of the nodes in the area containing that level one node; and

forwarding the multicast message from the level two node receiving the multicast messages to one or more level two nodes on the network.

4. A first communication node for use in a communication network also including other communication nodes to form a group of communication nodes interconnected by a plurality of communication links, said first communication node comprising:

means for selecting a set of said communication links, in common with the other communication nodes in the group, which form a multicast pathway having no loops and which permits any communication nodes in the group to send a message along the links in said multicast pathway to any other of said communication nodes in the group;

means for receiving a multicast message from the ones of said set of links which are coupled to said first communication node; and

means, coupled to said receiving means, for forwarding the received multicast message along every one of the links in said set, if any, which is connected to said first communication ode except the link over which the message was received.

5. A first communication node according to claim 4, wherein the selecting means includes

means for designating one of the communication nodes in the group, in common with the other nodes in the group, as a root node; and

means for choosing, as said multicast pathway, a spanning tree for the nodes in the group, said spanning tree being determined using said root node as the root of said spanning tree.

6. The first communication node of claim 4, further comprising:

a node memory including

a program memory for storing sequences of instructions,

a working memory for temporarily storing data and messages to be forwarded, and

a forwarding data base storing a list of links over which the multicast messages will be forwarded.

* * * * *

US006438705B1

(12) **United States Patent**
Chao et al.

(10) **Patent No.: US 6,438,705 B1**
(45) **Date of Patent: Aug. 20, 2002**

(54) **METHOD AND APPARATUS FOR BUILDING AND MANAGING MULTI-CLUSTERED COMPUTER SYSTEMS**

(75) Inventors: **Ching-Yun Chao**; **Patrick M. Goal**; **Richard James McCarty**, all of Austin, TX (US)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/240,494**

(22) Filed: **Jan. 29, 1999**

(51) **Int. Cl.**[7] **H02H 3/05**
(52) **U.S. Cl.** **714/4**; 709/226
(58) **Field of Search** 714/4, 1, 47, 11; 709/213, 223, 226, 249, 224; 712/13

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,964,838 A * 10/1999 Cheung et al. 709/224
6,003,075 A * 12/1999 Arendt et al. 709/221
6,154,765 A * 11/2000 Hart 709/201
6,249,879 B1 * 6/2001 Walker et al. 714/11
6,311,217 B1 * 10/2001 Ehlinger et al. 709/226

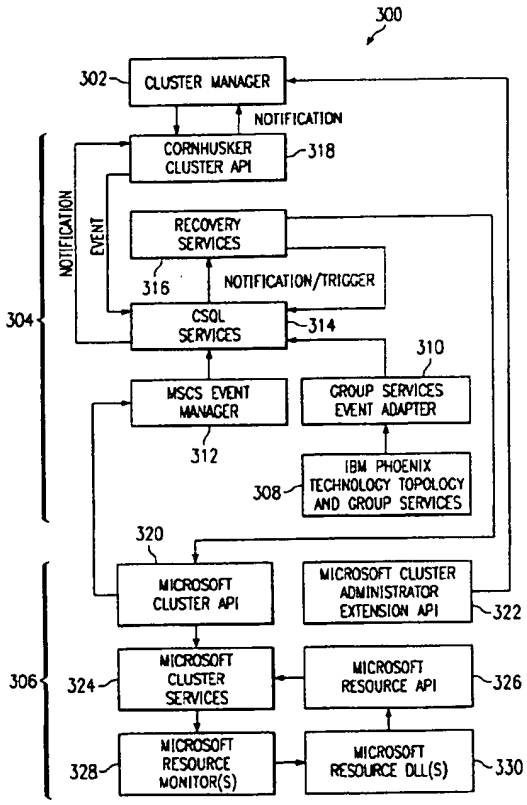
* cited by examiner

Primary Examiner—Robert Beausoleil
Assistant Examiner—Rita A Ziemer
(74) *Attorney, Agent, or Firm*—Duke W. Yee; Jeffrey S. LaBaw; Stephen J. Walder, Jr.

(57) **ABSTRACT**

One application of clustered computer systems is to support failover of applications and shared resources. Another is to support scalable or fault-tolerant distributed applications. The present invention utilizes a higher-level clustering mechanism (a multi-cluster) overlayed on top of multiple underlying clusters (subclusters) to extend their capabilities. In the described embodiment, subclusters supporting application and shared resource failover across a smaller number of nodes is overlayed with a multi-cluster supporting a larger number of nodes. The multi-cluster manages cluster-level communication among nodes, and the subclusters directly manage only the shared device and application resources which they are permitted to control. It is possible to move resources between nodes which reside in different subclusters. The multi-cluster layer also externalizes interfaces similar to those of the subclusters, providing application compatibility.

43 Claims, 9 Drawing Sheets



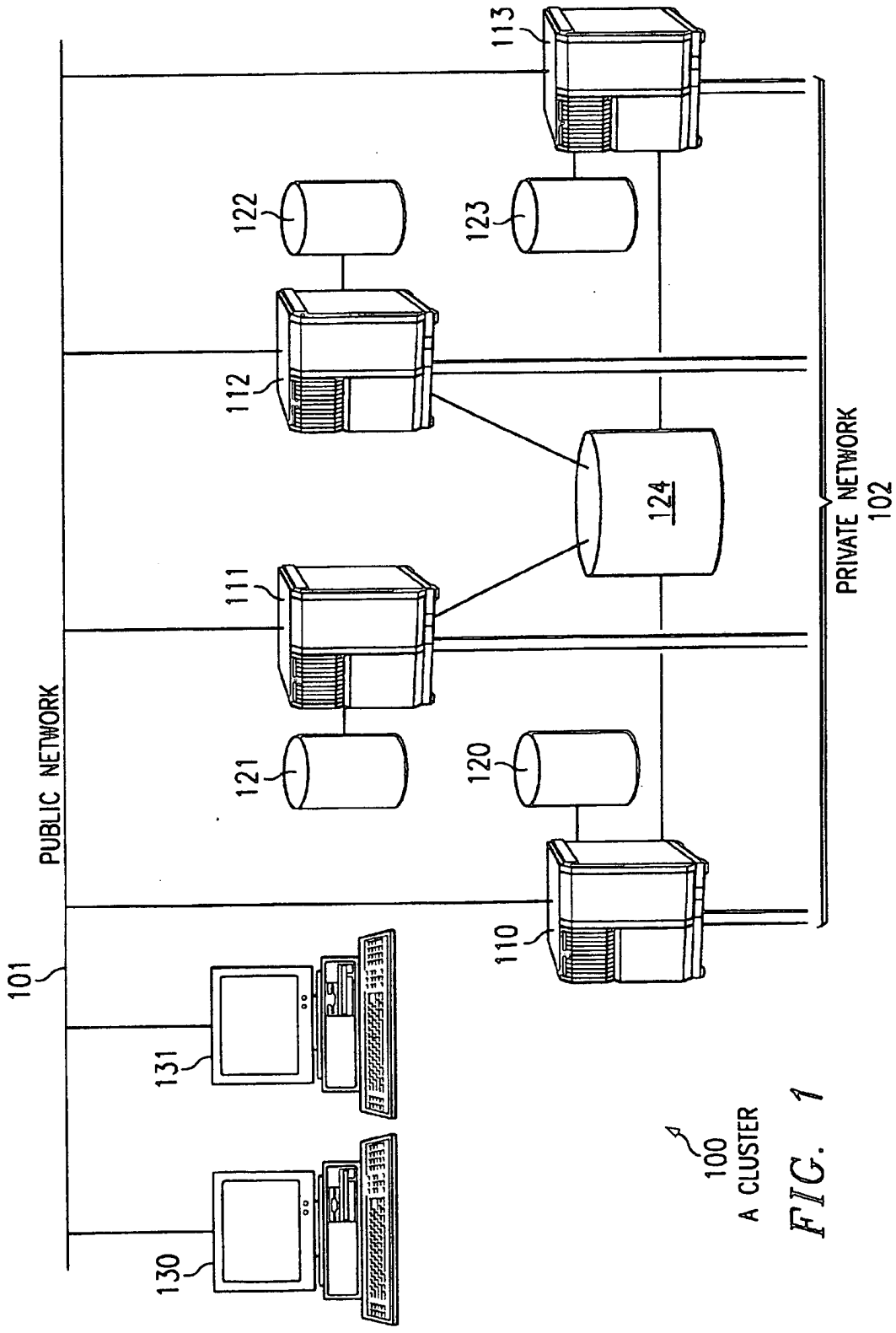

113
122
123
112
124
111
121
120
110
PUBLIC NETWORK
101
PRIVATE NETWORK
102
131
130
100
A CLUSTER

FIG. 1

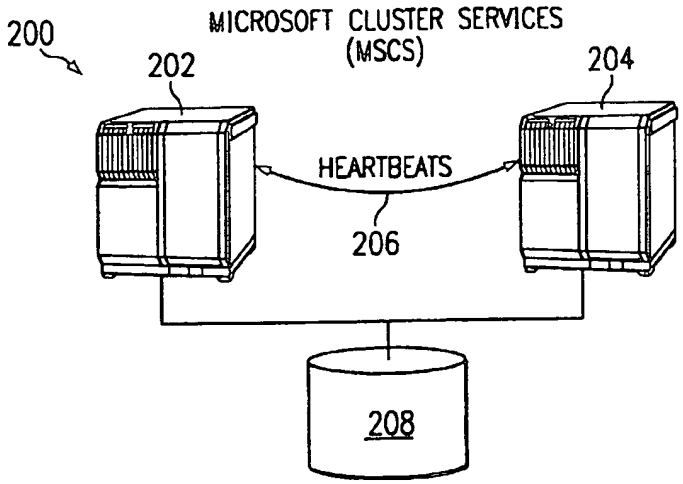


FIG. 2a
(PRIOR ART)

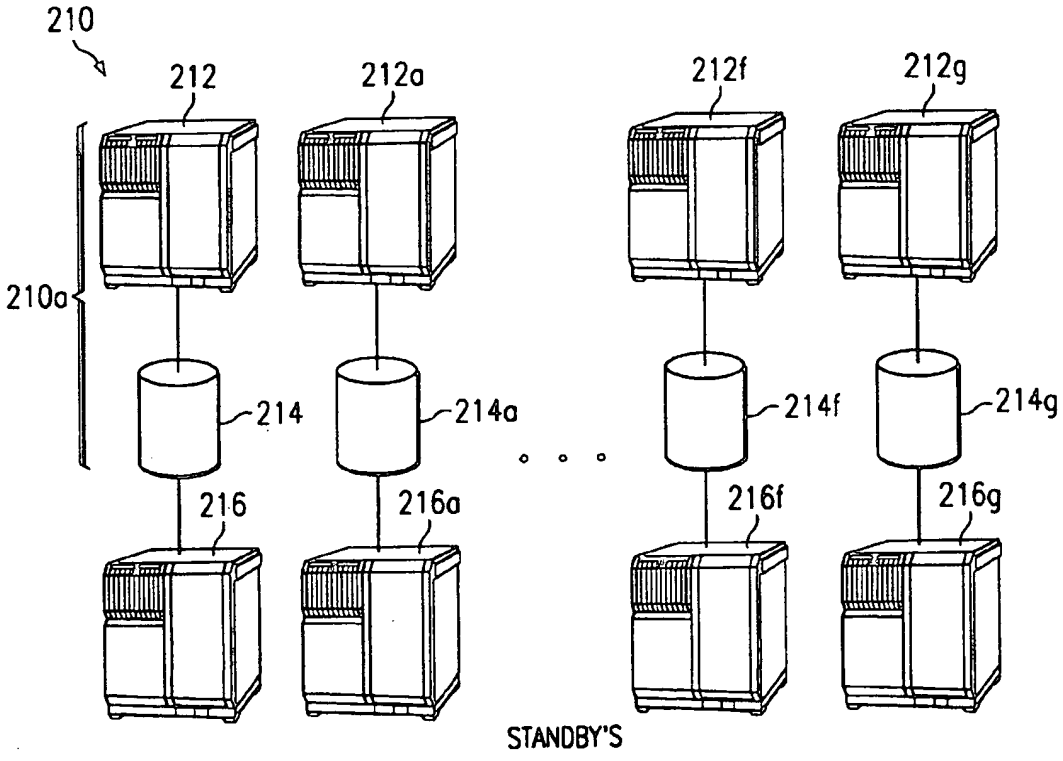


FIG. 2b
(PRIOR ART)

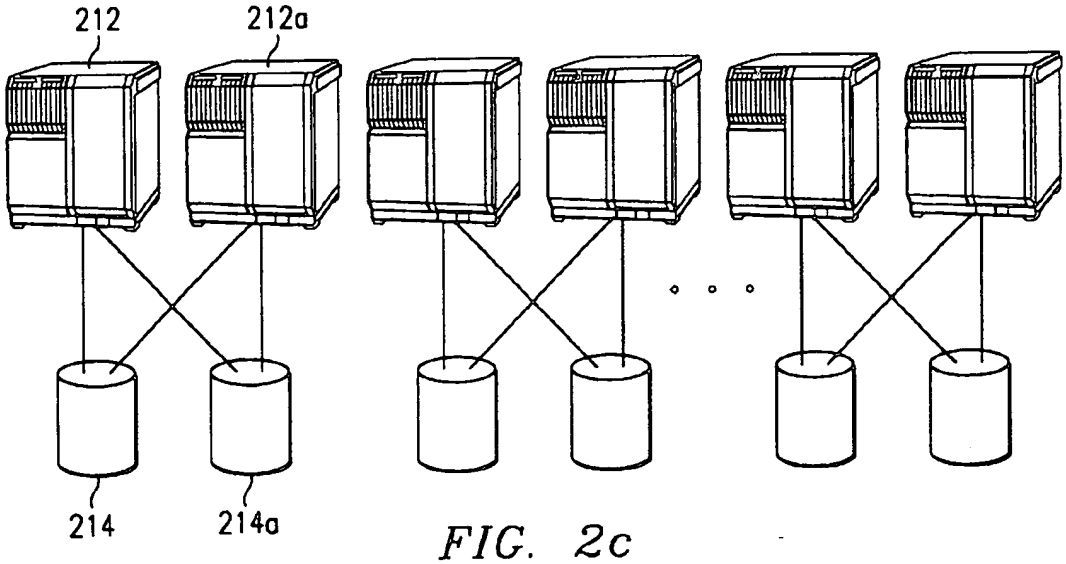


FIG. 2c
(PRIOR ART)

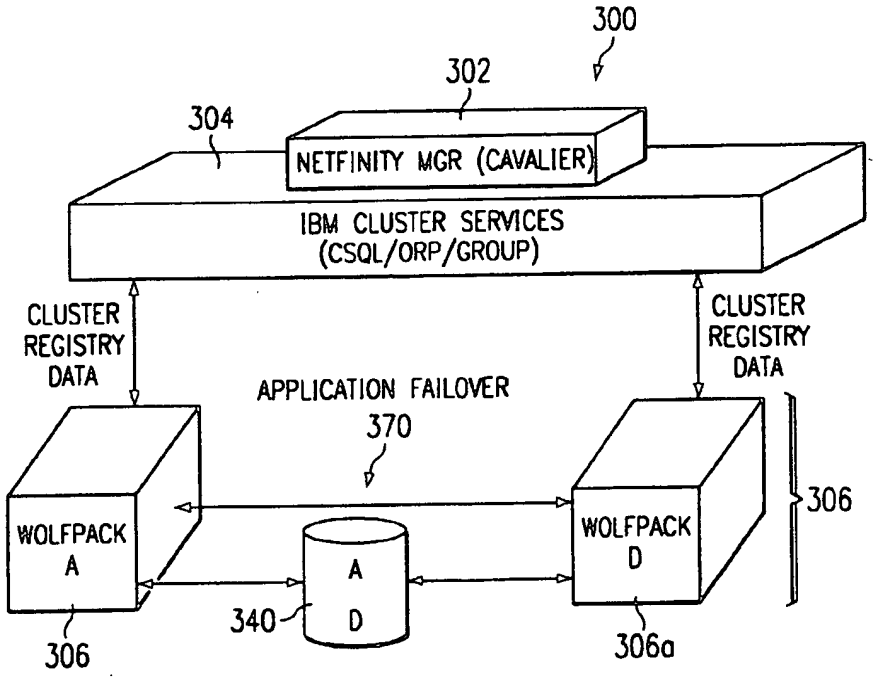


FIG. 3

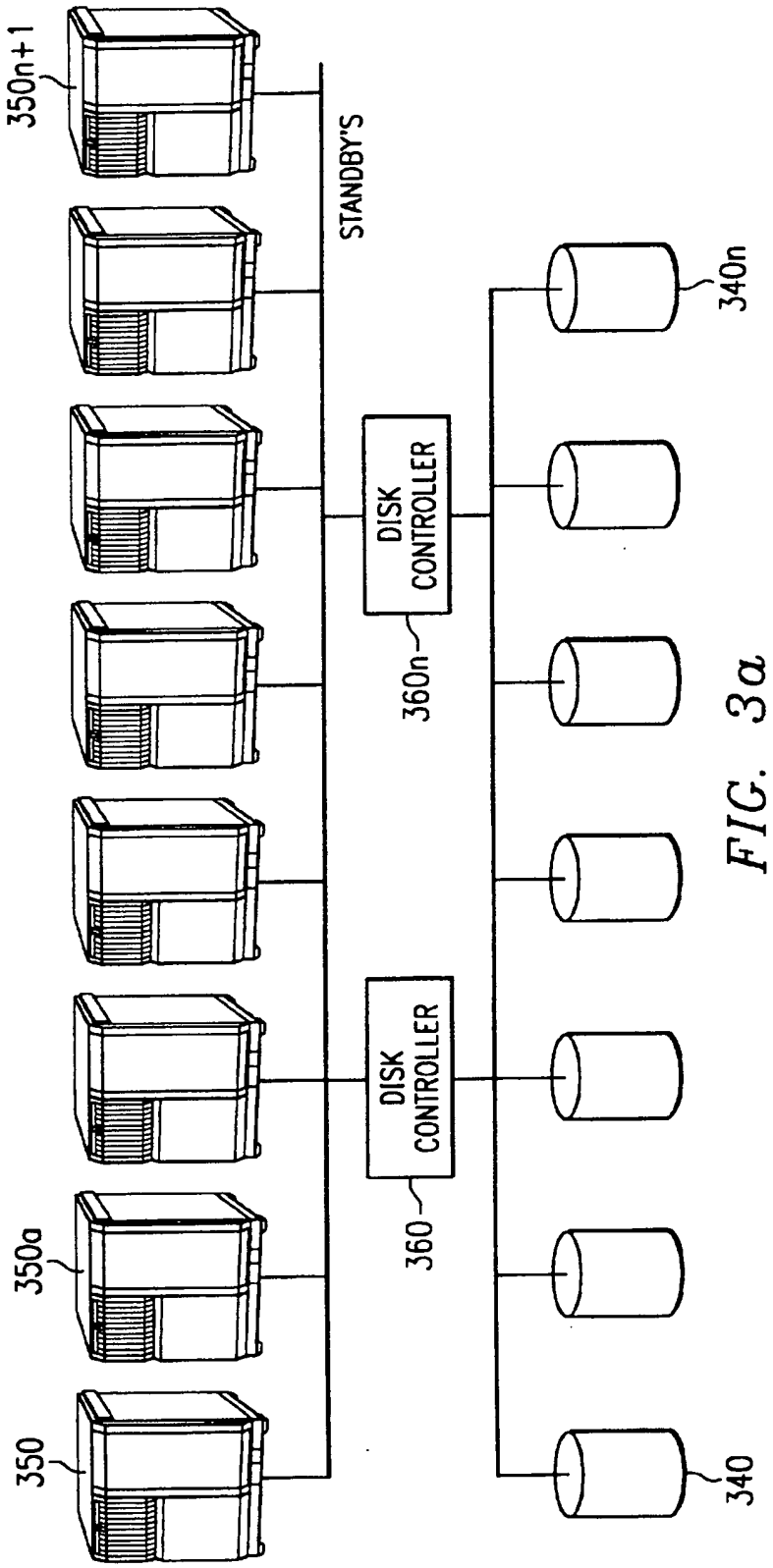

350n+1
STANDBY'S
DISK CONTROLLER
360n
340n
DISK CONTROLLER
360
350a
350
340

FIG. 3a

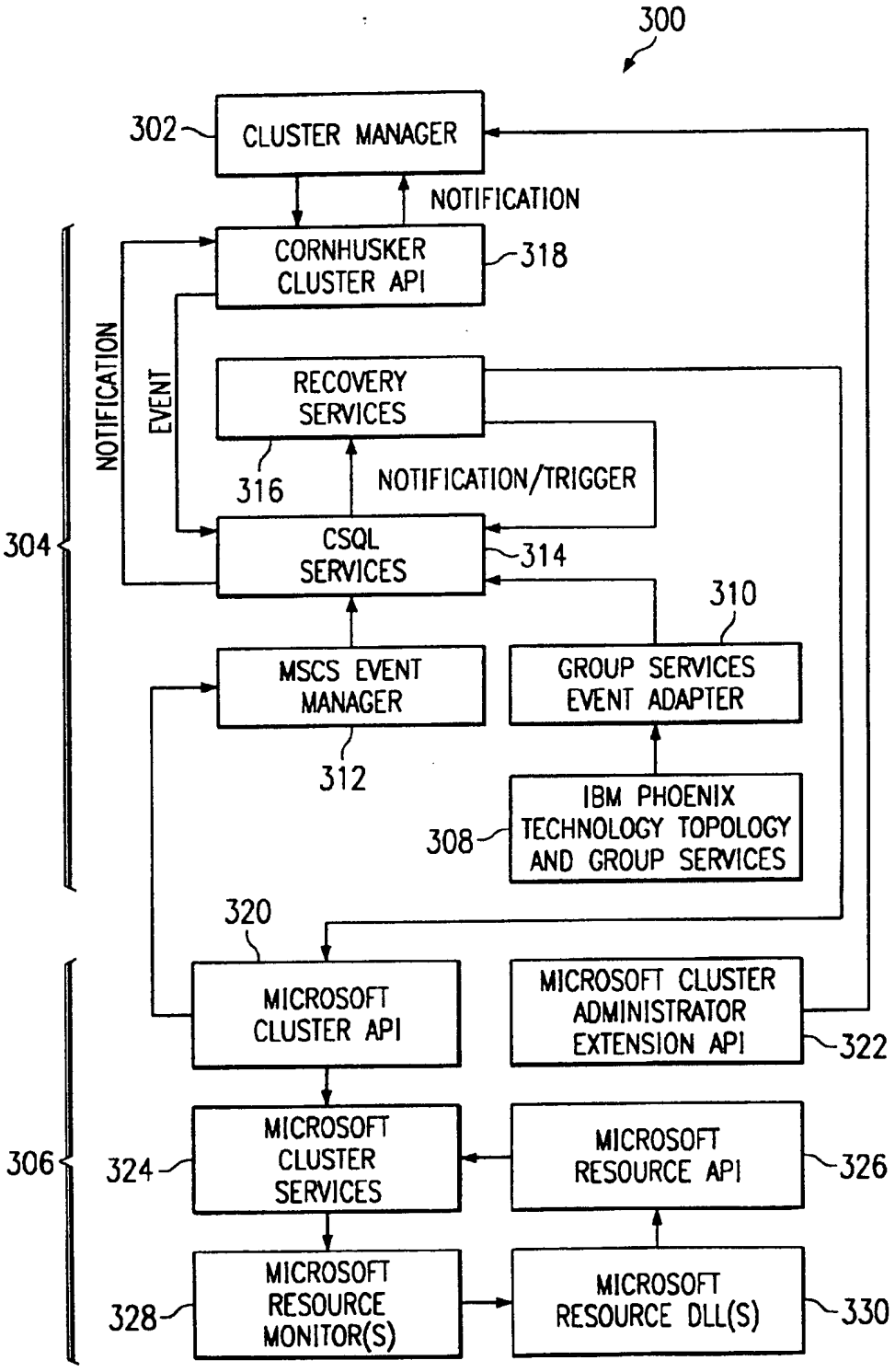

300
302
CLUSTER MANAGER
NOTIFICATION
CORNHUSKER CLUSTER API
318
NOTIFICATION
EVENT
RECOVERY SERVICES
316
NOTIFICATION/TRIGGER
304
CSQL SERVICES
314
310
MSCS EVENT MANAGER
GROUP SERVICES EVENT ADAPTER
312
IBM PHOENIX TECHNOLOGY TOPOLOGY AND GROUP SERVICES
308
320
MICROSOFT CLUSTER API
MICROSOFT CLUSTER ADMINISTRATOR EXTENSION API
322
306
324
MICROSOFT CLUSTER SERVICES
MICROSOFT RESOURCE API
326
328
MICROSOFT RESOURCE MONITOR(S)
MICROSOFT RESOURCE DLL(S)
330

FIG. 3b

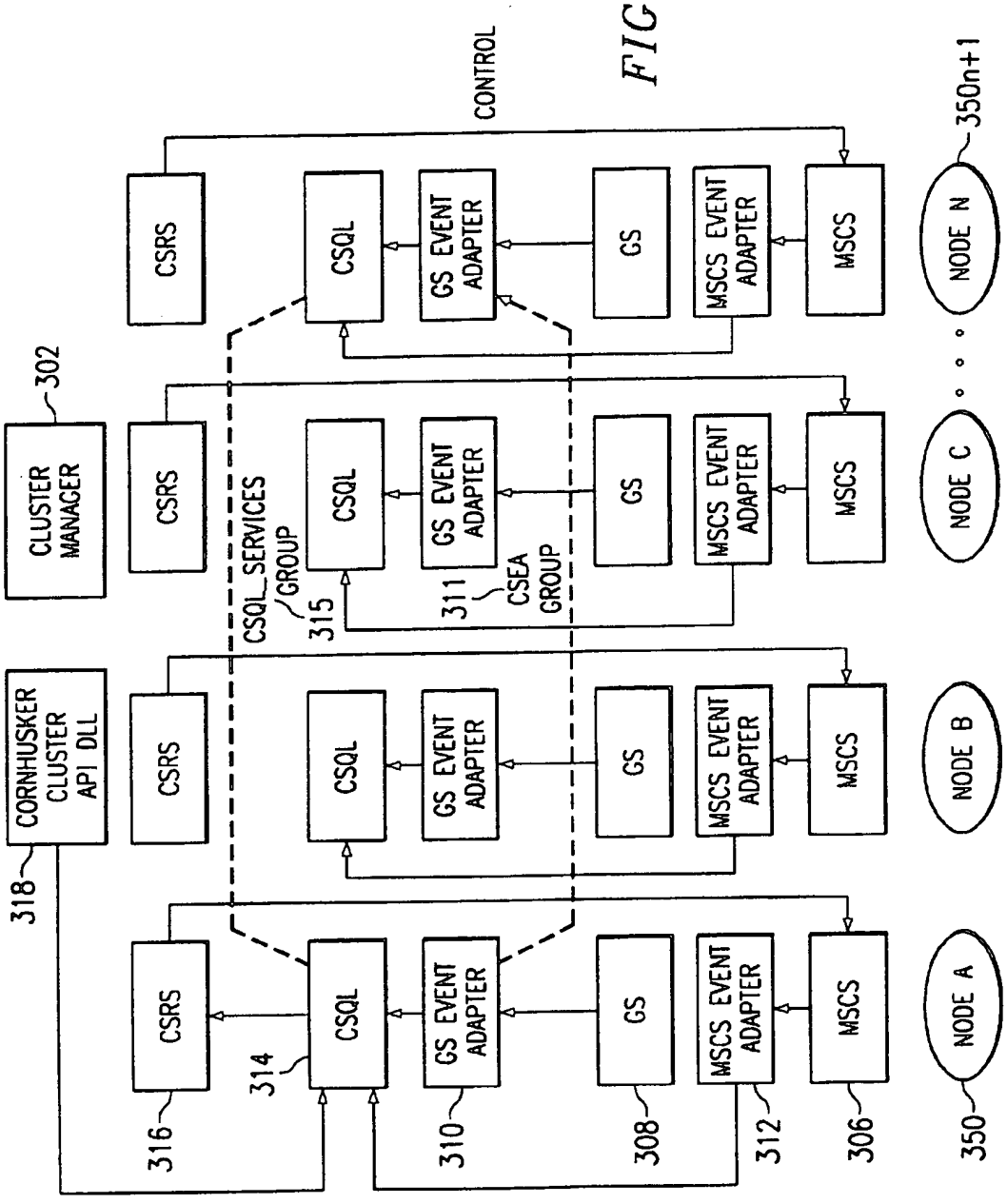

CONTROL
CSRS
CSQL
GS EVENT ADAPTER
GS
MSCS EVENT ADAPTER
MSCS
NODE N
350n+1
302
CLUSTER MANAGER
CSQL_SERVICES GROUP
315
311
CSEA GROUP
NODE C
CORNHUSKER CLUSTER API DLL
318
NODE B
314
316
310
308
312
306
NODE A
350

FIG. 3c

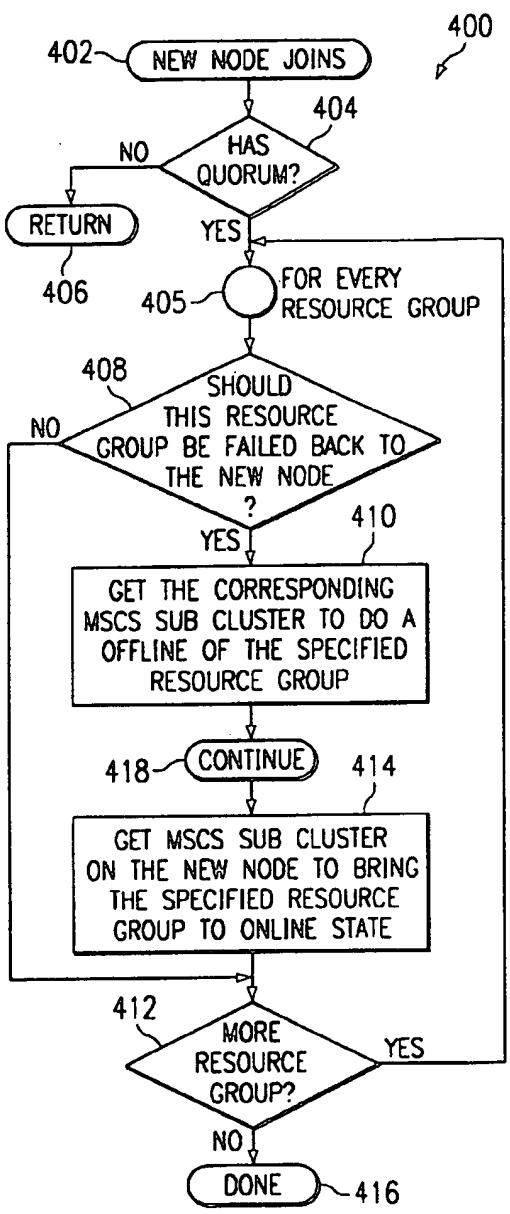

400
402
NEW NODE JOINS
404
HAS QUORUM?
NO
RETURN
406
YES
405
FOR EVERY RESOURCE GROUP
408
SHOULD THIS RESOURCE GROUP BE FAILED BACK TO THE NEW NODE ?
NO
YES
410
GET THE CORRESPONDING MSCS SUB CLUSTER TO DO A OFFLINE OF THE SPECIFIED RESOURCE GROUP
418
CONTINUE
414
GET MSCS SUB CLUSTER ON THE NEW NODE TO BRING THE SPECIFIED RESOURCE GROUP TO ONLINE STATE
412
MORE RESOURCE GROUP?
YES
NO
DONE
416

FIG. 4

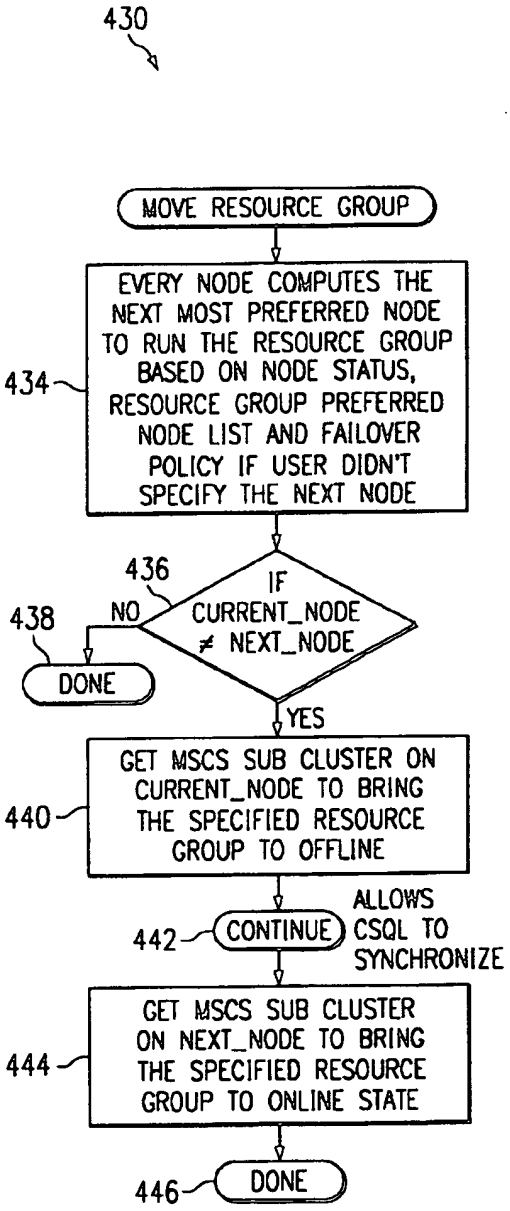

430
MOVE RESOURCE GROUP
EVERY NODE COMPUTES THE NEXT MOST PREFERRED NODE TO RUN THE RESOURCE GROUP BASED ON NODE STATUS, RESOURCE GROUP PREFERRED NODE LIST AND FAILOVER POLICY IF USER DIDN'T SPECIFY THE NEXT NODE
434
436
IF CURRENT_NODE ≠ NEXT_NODE
NO
438
DONE
YES
GET MSCS SUB CLUSTER ON CURRENT_NODE TO BRING THE SPECIFIED RESOURCE GROUP TO OFFLINE
440
CONTINUE
442
ALLOWS CSQL TO SYNCHRONIZE
GET MSCS SUB CLUSTER ON NEXT_NODE TO BRING THE SPECIFIED RESOURCE GROUP TO ONLINE STATE
444
DONE
446

FIG. 4a

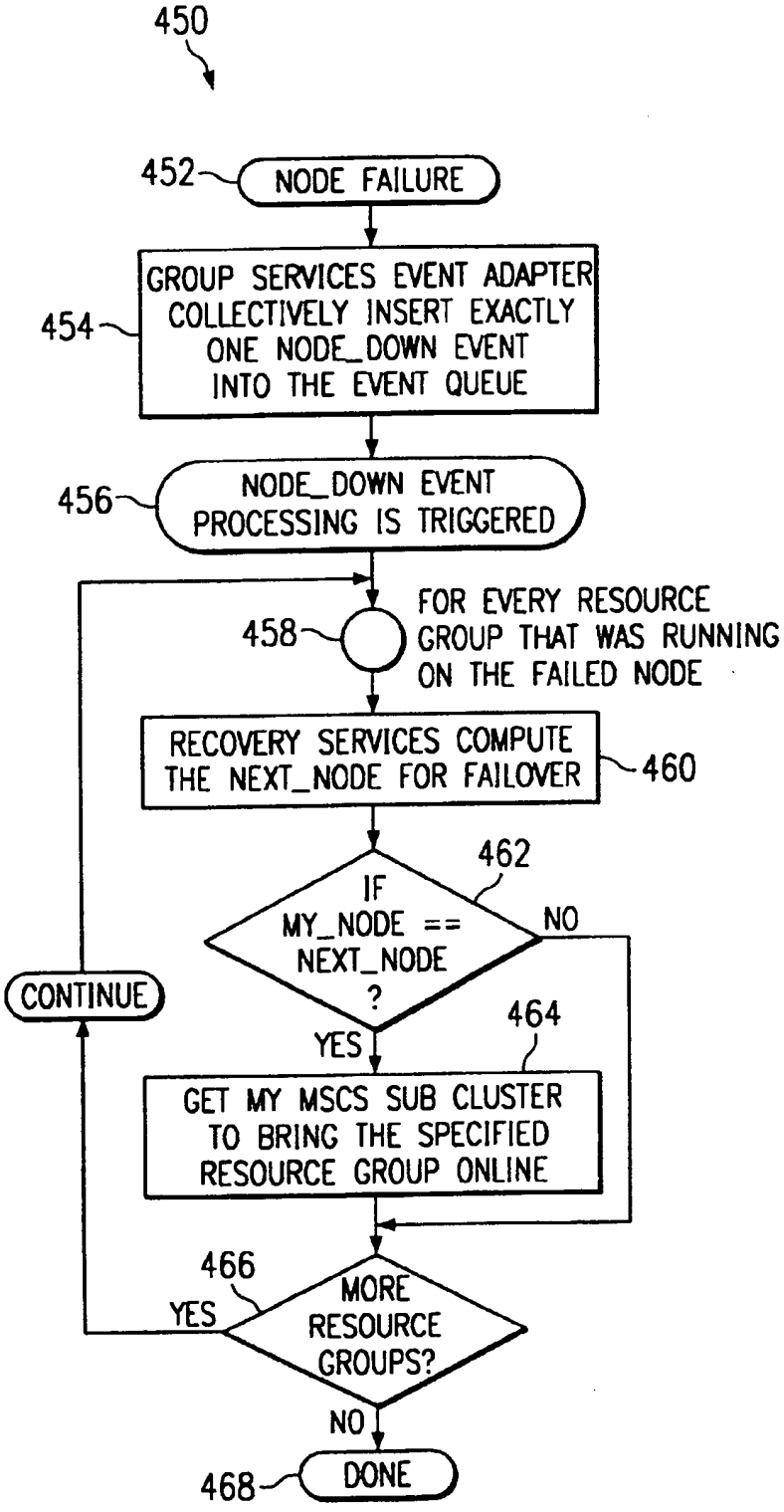

450
452
NODE FAILURE
454
GROUP SERVICES EVENT ADAPTER COLLECTIVELY INSERT EXACTLY ONE NODE_DOWN EVENT INTO THE EVENT QUEUE
456
NODE_DOWN EVENT PROCESSING IS TRIGGERED
458
FOR EVERY RESOURCE GROUP THAT WAS RUNNING ON THE FAILED NODE
RECOVERY SERVICES COMPUTE THE NEXT_NODE FOR FAILOVER
460
462
IF MY_NODE == NEXT_NODE ?
NO
YES
464
GET MY MSCS SUB CLUSTER TO BRING THE SPECIFIED RESOURCE GROUP ONLINE
CONTINUE
466
MORE RESOURCE GROUPS?
YES
NO
468
DONE

FIG. 4b

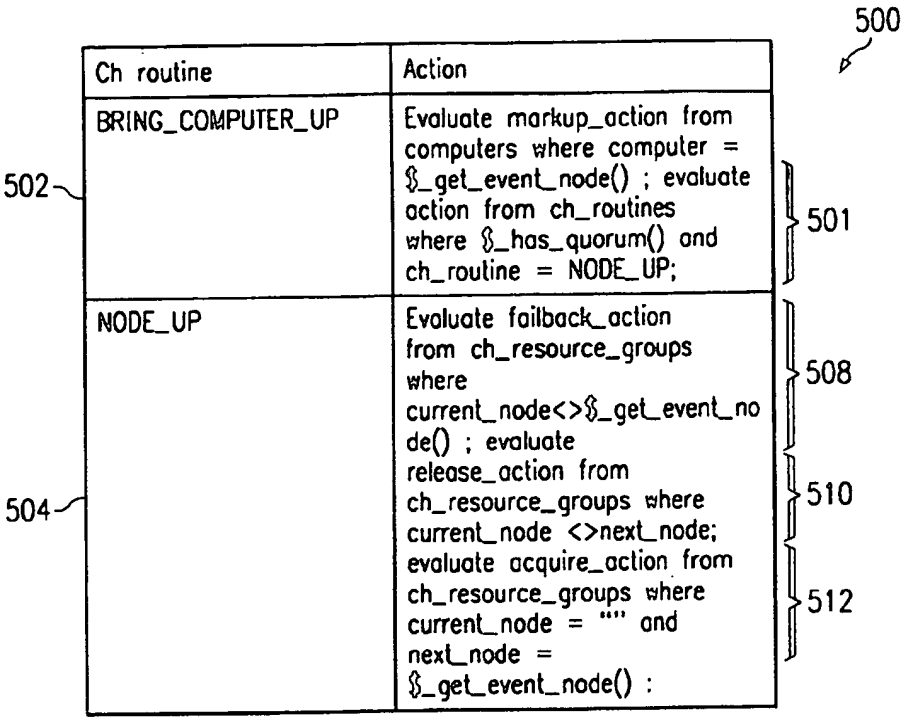


FIG. 5

600

RESOURCE GROUP TABLE (CLASS) CLASS METHODS

ch_resource_group	A_sample_resource_group
failover_policy	cascading
failback_policy	autohoming
failback_action	update ch_resource_groups set next_node = $_failback_node() where ch_resource_group = this ch_resource_group;
release_action	execute $_resource_group_offline() ;
acquire_action	execute $_resource_group_online() ;
current_node	
next_node	

FIG. 6

METHOD AND APPARATUS FOR BUILDING AND MANAGING MULTI-CLUSTERED COMPUTER SYSTEMS

CROSS REFERENCE TO RELATED APPLICATIONS

The present invention is related to application entitled Method And Apparatus For Building And Managing Multi-Clustered Computer Systems, Ser. No. 09/181,825, filed Oct. 29, 1998, assigned to the same assignee, and incorporated herein by reference.

BACKGROUND OF THE INVENTION

1. Technical Field

The present invention relates generally to a distributed data processing system and in particular to a method and apparatus for managing a server system within a distributed data processing system. Still more particularly, the present invention relates to a method and apparatus for managing a clustered computer system.

2. Description of Related Art

A clustered computer system is a type of parallel or distributed system that consists of a collection of interconnected whole computers and is used as a single, unified computing resource. The term "whole computer" in the above definition is meant to indicate the normal combination of elements making up a stand-alone, usable computer: one or more processors, an acceptable amount of memory, input/output facilities, and an operating system. Another distinction between clusters and traditional distributed systems concerns the relationship between the parts. Modern distributed systems use an underlying communication layer that is peer-to-peer. There is no intrinsic hierarchy or other structure, just a flat list of communicating entities. At a higher level of abstraction, however, they are popularly organized into a client-server paradigm. This results in a valuable reduction in system complexity. Clusters typically have a peer-to-peer relationship.

There are three technical trends to explain the popularity of clustering. First, microprocessors are increasingly fast. The faster microprocessors become, the less important massively parallel systems become. It is no longer necessary to use super-computers or aggregations of thousands of microprocessors to achieve suitably fast results. A second trend that has increased the popularity of clustered computer systems is the increase in high-speed communications between computers. A cluster computer system is also referred to as a "cluster". The introduction of such standardized communication facilities as Fibre Channel Standard (FCS), Asynchronous Transmission Mode (ATM), the Scalable Coherent Interconnect (SCI), and the switched Gigabit Ethernet are raising inter-computer bandwidth from 10 Mbits/second through hundreds of Mbytes/second and even Gigabytes per second. Finally, standard tools have been developed for distributed computing. The requirements of distributed computing have produced a collection of software tools that can be adapted to managing clusters of machines. Some, such as the Internet communication protocol suite (called TCP/IP and UDP/IP) are so common as to be ubiquitous de facto standards. High level facilities built on the base, such as Intranets, the Internet and the World Wide Web, are similarly becoming ubiquitous. In addition, other tool sets for multi-sense administration have become common. Together, these are an effective base to tap into for creating cluster software.

In addition to these three technological trends, there is a growing market for computer clusters. In essence, the market is asking for highly reliable computing. Another way of stating this is that the computer networks must have "high availability." For example, if the computer is used to host a web-site, its usage is not necessarily limited to normal business hours. In other words, the computer may be accessed around the clock, for every day of the year. There is no safe time to shut down to do repairs. Instead, a clustered computer system is useful because if one computer in the cluster shuts down, the others in the cluster automatically assume its responsibilities until it can be repaired. There is no down-time exhibited or detected by users.

Businesses need "high availability" for other reasons as well. For example, business-to-business intranet use involves connecting businesses to subcontractors or vendors. If the intranet's file servers go down, work by multiple companies is strongly affected. If a business has a mobile workforce, that workforce must be able to connect with the office to download information and messages. If the office's server goes down, the effectiveness of that work force is diminished.

A computer system is highly available when no replaceable piece is a single point of failure, and overall, it is sufficiently reliable that one can repair a broken part before something else breaks. The basic technique used in cluster to achieve high availability is failover. The concept is simple enough: one computer (A) watches over another computer (B); if B dies, A takes over B's work. Thus, failover involves moving "resources" from one node to another. A node is another term for a computer. Many different kinds of things are potentially involved: physical disk ownership, logical disk volumes, IP addresses, application processes, subsystems, print queues, collection of cluster-wide locks in a shared-data system, and so on.

Resources depend on one another. The relationship matters because, for example, it will not help to move an application to one node when the data it uses is moved to another. Actually it will not even help to move them both to the same node if the application is started before the necessary disk volumes are mounted.

In modern cluster systems such as IBM HACMP and Microsoft "Wolfpack", the resource relationship information is maintained in a cluster-wide data file. Resources that depend upon one another are organized as a resource group and are stored as a hierarchy in that data file. A resource group is the basic unit of failover.

With reference now to the figures, and in particular with reference to FIG. **1**, a pictorial representation of a distributed data processing system in which the present invention may be implemented is depicted.

Distributed data processing system **100** is a network of computers in which the present invention may be implemented. Distributed data processing system **100** contains one or more public networks **101**, which is the medium used to provide communications links between various devices, client computers, and server computers connected within distributed data processing system **100**. Network **100** may include permanent connections, such as Token Ring, Ethernet, 100 Mb Ethernet, Gigabit Ethernet, FDDI ring, ATM, and high speed switch, or temporary connections made through telephone connections. Client computers **130** and **131** communicate to server computers **110**, **111**, **112**, and **113** via public network **101**.

Distributed data processing system **100** optionally has its own private communications networks **102**. Communications on network **102** can be done through a number of means: standard networks just as in **101**, shared memory,

shared disks, or anything else. In the depicted example, a number of servers **110**, **111**, **112**, and **113** are connected both through the public network **101** as well as private networks **102**. Those servers make use of the private network **102** to reduce the communication overhead resulting from heartbeating each other and running membership and n-phase commit protocols.

In the depicted example, all servers are connected to a shared disk storage device **124**, preferably a Redundant Array of Independent Disks (RAID) device for better reliability, which is used to store user application data. Data are made highly available in that when a server fails, the shared disk partition and logical disk volume can be failed over to another node so that data will continue to be available. The shared disk interconnection can be Small Computer System Interface (SCSI) bus, Fibre Channel, and International Business Machines Serial Storage Architecture (IBM SSA). Alternatively, each server machine can also have local data storage device **120**, **121**, **122**, and **123**. FIG. **1** is intended as an example, and not as an architectural limitation for the processes of the present invention.

Referring to FIG. *2a*, cluster computer system **200** using Microsoft Cluster Services (MSCS) is designed to provide high availability for NT Server-based applications. The initial MSCS supports failover capability in a two-node **202**, **204**, shared disk **208** cluster.

Each MSCS cluster consists of one or two nodes.

Each node runs its own copy of Microsoft Cluster Services. Each node also has one or more Resource Monitors that interact with the Microsoft Cluster Services. These monitors keep the Microsoft Cluster Services "informed" as to the status of individual resources. If necessary, the resource Monitor can manipulate individual resources through the use of Resource DLLs. When a resource fails, Microsoft Cluster Services will either restart it on the local node or move the resource group to the other node, depending on the resource restart policy and the resource group failover policy and cluster status.

The two nodes in a MSCS cluster heartbeat **206** each other. When one node fails, i.e., fails to send heartbeat signal to the other node, all its resource groups will be restarted on the remaining node. When a cluster node is booted, the cluster services are automatically started under the control of the event processor. In addition to its normal role of dispatching events to other components, the event processor performs initialization and then tells the node manager, also called the membership manager, to join or create the cluster.

The node manager's normal job is to create a consistent view of the state of cluster membership, using heartbeat exchange with the other node managers. It knows who they are from information kept in its copy of the cluster configuration database, which is actually part of the Windows NT registry (but updated differently). The node manager initially attempts to contact the other node, if it succeeds, it tries to join the cluster, providing authentication (password, cluster name, its own identification, and so on). If there is an existing cluster and for some reason our new node's attempt to join is rejected, then the node and the cluster services located on that node will shutdown.

However, if the other node does not respond to a node's requests to join up, the node manager tries to start up a new cluster. To do that, it uses a special resource, specified like all resources in a configuration database, called the quorum resource. There is exactly one quorum resource in every cluster. It's actually a disk; if it is, it's very preferable to have it mirrored or otherwise fault-tolerant, as well as multiported with redundant adapter attachments, since otherwise it will be a single point of failure for the cluster. The device used as a quorum resource can be anything with three properties: it can store data durably (across failure); the other cluster node can get at it; and it can be forcibly acquired by one node to the exclusion of all others. SCSI and other disk protocols like SSA and FC-AL allow for exactly this operation.

The quorum resource is effectively a global control lock for the cluster. The node that successfully seizes the quorum resources uniquely defines the cluster. The other node must join with that one to become part of the cluster. This prohibits the problem of a partitioned cluster. It is possible for internal cluster communication to fail in a way that breaks the cluster into two partitions that cannot communicate with each other. The node that controls the quorum resource is the cluster, and there is no other cluster.

Once a node joins or forms a cluster, the next thing it does is update its configuration database to reflect any changes that were made while it was away. The configuration database manager can do this because, of course, changes to that database must follow transactional semantics consistently across all the nodes and, in this case, that involves keeping a log of all changes stored on the quorum device. After processing the quorum resource's log, the new node will begin to acquire resources. These can be disks, IP names, network names, applications, or anything else that can be either off-line or on-line. They are all listed in the configuration database, along with the nodes they would prefer to run on, the nodes they can run on (some may not connect to the right disks or networks), their relationship to each other, and everything else about them. Resources are typically formed into and managed as resource groups. For example, an IP address, a file share (sharable unit of a file system), and a logical volume might be the key elements of a resource group that provides a network file system to clients. Dependencies are tracked, and no resource can be part of more than one resource group, so sharing of resources by two applications is prohibited unless those two applications are in the same resource group.

The new node's failover manager is called upon to figure out what resources should move (failover) to the new node. It does this by negotiating with the other node's failover managers, using information like the resources, preferred nodes. When they have come to a collective decision, any resource groups that should move to this one from the other node are taken off-line on that node; when that is finished, the Resource Manager begins bringing them on-line on the new node.

Every major vendor of database software has a version of their database that operates across multiple NT Servers. IBM DB2 Extended Enterprise Edition runs on 32 nodes. IBM PC Company has shipped a 6-node PC Server system that runs Oracle Parallel Servers. There is no adequate system clustering software for those larger clusters.

In a 6-node Oracle Parallel Server system, those six nodes share the common disk storage. Oracle uses its own clustering features to manage resources and to perform load balancing and failure recovery. Customers that run their own application software on those clusters need system clustering features to make their applications highly available.

Referring to FIG. 2B, DB2 typically uses a shared nothing architecture **210** where each node **212** has its own data storage **214**. Databases are partitioned and database requests are distributed to all nodes for parallel processing. To be highly available, DB2 uses failover functionality from sys-

tem clustering. Since MSCS supports only two nodes, DB2 must either allocate a standby node **216** for each node **212** as shown. Alternatively, DB2 can allow mutual failover between each pair of MSCS nodes as shown in FIG. **2***c*. In other words, two nodes **212**, **212***a* are mutually coupled to two data storages **214**, **214***a*. The former doubles the cost of a system and the latter suffers performance degradation when a node fails. Because database access is distributed to all nodes and are processed in parallel, the node that runs both its DB2 instance and the failed over instance becomes the performance bottleneck. In other words, if node **212***a* fails, then node **212** assumes its responsibilities and accesses data on both data storages, but runs its tasks in parallel.

Therefore, it would be advantageous to have an improved method and apparatus for managing a cluster computer system. Such an improvement should allow support of a failover from one node to another node chosen from a group of many nodes.

SUMMARY OF THE INVENTION

The present invention provides a method and apparatus for managing clustered computer systems and extends clustering to very large clusters by providing a mechanism to manage a number of cluster computer systems, also referred to as "clusters". In particular, the present invention detects an initiation of a restart of a cluster computer system within a number of cluster computer systems. The initiation of the restart of the cluster computer system will cause the cluster computer system to restart in a selected state. In addition, this cluster computer system includes one or more resources. In response to a determination that one or more of the resources within the cluster computer system that is being restarted is presently on-line on another node within the cluster computer system, the restart of those resources will be prevented.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

FIG. **1** is a pictorial representation of a distributed data processing system in which the present invention may be implemented;

FIGS. **2***a*, **2***b*, and **2***c* provide illustrations of the Microsoft MSCS product and its limitations in implementation;

FIGS. **3**, **3***a*, **3***b* and **3***c* illustrate the present invention and illustrate its implementation across multiple clusters such as MSCS clusters;

FIGS. **4**, **4***a*, and **4***b* are flowcharts of underlying methods used by the present invention to control multiple clusters; and

FIGS. **5** and **6** are SQL tables containing example configuration, status, and event processing rules used with the present invention.

DETAILED DESCRIPTION OF THE DRAWINGS

The present invention extends the Microsoft Cluster Manager functionality to manage the larger cluster but otherwise preserves its ease-of-use characteristics. When discussed in this application, a "multi-cluster" refers to a cluster of two or more cluster computer systems.

Further, the present cluster system supports resource group failover among any two nodes in a larger cluster of two or more nodes. The present system also preserves the application state information across the entire cluster in the case of failure events. Also, the present system does not require change implementation of currently available cluster computer system products. For example, with respect to MSCS, the mechanism of the present invention does not require Microsoft and application vendors to make any modification to their present clustering code in order to run in this system's environment. Instead, the present system provides an implementation of the MSCS Cluster API DLL that is binary compatible with the MSCS Cluster API DLL.

A multi-cluster normally contains more than one underlying cluster. The present invention provides a cluster manager that can configure a cluster with multiple MSCS clusters within. Resources in a multi-cluster are managed by each individual cluster under the supervision of Cluster Services. No need exists to modify the Microsoft Resource API and the Microsoft Cluster Administrator extension API. The Cluster Manager can use any Cluster Administrator Extension DLL that is developed for MSCS as it is without modification.

Applications, whether they are enhanced for MSCS or not, can readily take advantage of system clustering features of the present invention. Instead of mutual failover between one pair of nodes, the present invention allows an application failover between any two nodes in a large cluster. The present invention allows a cluster to grow in size by adding an MSCS cluster either with a pair of nodes or a single node. The fact that the present invention can support a three node cluster is very attractive to many customers who want to further improve availability of their mission critical applications over a two node cluster.

Applications such as DB2 Extended Enterprise Edition that use MSCS can readily take advantage of multi-cluster system clustering features. DB2/EEE exploits MSCS features by dividing nodes into pairs and allows mutual failover between each pair of nodes as discussed above in reference to FIG. **2***c*. The present invention can either improve DB2 availability by supporting N-way fail-over or improve DB2 performance characteristics by supporting N+1 model with one standby node. In the most common event of a single node failure, DB2/EEE instance on the failed node will be restarted on the standby node and maintain the same performance in the N+1 mode. System management policy and recovery services are expressed in a high-level language that can be modified easily to tailor to special requirements from application vendors. For example, this allows DB2/EEE to be integrated with a multi-cluster better than with a MSCS cluster.

It must be understood that the present invention can be used over any cluster service program. While the depicted example illustrates MSCS clusters within a multi-cluster, the processes, mechanisms, and instruction of the present invention may be applied to managing clusters of all types. The present invention is not limited in any way to use over that particular product. For example, the present invention may be applied to heterogeneous multi-clusters.

With reference now to FIG. **3**, a pictorial representation of a distributed data processing system in which the present invention may be implemented is depicted. The software **300** shown in FIGS. **3**, **3***b*, and **3***c* can be implemented on the hardware shown in FIG. **3***a*. The process for the multi-cluster software illustrated herein can scale to larger sizes easily. For example, FIG. **3***a* shows an eight-node

configuration, wherein each node **350** is coupled to a storage element **340** by disk controllers **360**. Cluster services **304** in FIG. **3** allows fail-over to be between any two nodes in this eight-node cluster. The use of the term "cluster services" is used herein to refer to the services provided by the present invention. The cluster services, such as cluster services **304**, is employed to control a cluster, such as a MSCS cluster. It can be used in both the Oracle cluster or a DB2 cluster discussed above. In the case when any of the seven nodes fails, the DB2 instance will be restarted on the eight node and the performance of the system will remain unchanged. This is called an N+1 failover model. Other configurations are also supported. For example, each node may run an active DB2 instance and backup the other seven nodes to maximize reliability. MSCS is used to perform resource management for a single node in the depicted example. Microsoft does not share its resource management APIs in Windows NT with outside vendors and there is no easy way for other vendors to perform resource management. Some vendors implemented their own device drivers and TCP/IP protocol stack. That results in incompatibility with the MSCS Cluster API and Resource API. The present invention uses MSCS to manage resources on a single node, and thus does not need to know the internal NT APIs. Again, while reference is made herein to the Microsoft cluster product, the present invention is in no way limited to use over that product. The present invention can be used over any cluster services program.

Referring to FIG. **3**, cluster services **304** controls MSCS **306** to bring a resource and a resource group on-line or off-line on a node **350**. Cluster services **304** is shown controlling the MSCS **306** and **306***a*, which are located on different nodes **350** and **350***a*. Cluster Services **304** causes MSCS **306** to bring resource group containing application **370** off-line and then cause MSCS **306***a* to bring that resource group on-line. Cluster services **304** is responsible for managing cluster node membership, heartbeat, inter-node communications, and for maintaining the consistency of cluster configuration database for all eight nodes. Cluster services also is responsible for event notification and processing. Cluster manager **302** provides a graphical user interface (GUI).

Cluster services **304** is substantially binary compatible with MSCS in this example. No modification is required to run any application in a multi-cluster if that application can run in an MSCS cluster. Cluster services supports all MSCS Cluster API, Resource API, and Administrator Extension API.

Referring to FIGS. **3***b* and **3***c* in a multi-cluster, each node runs a copy of Cluster Services. When a node **350** is booted, cluster services **304** is started automatically. The MSCS cluster services **306** is then started by cluster services **304**. In this document, we will refer to those MSCS clusters within a multi-cluster as MSCS subclusters. The configuration information in a multi-cluster configuration database is a super set of the information in each MSCS subcluster. All resources and resources groups are defined in multi-cluster configuration database and in appropriate MSCS subclusters. When an MSCS subcluster services is started, all resources and resources groups except the default Cluster Group are left in off-line state. Cluster services **304** on a new node determines collectively through CSQL Services group **315** with cluster services instances on all other nodes which resource groups should be started on that node. It then invokes the MSCS cluster services API to bring those resource groups to an on-line state.

Each MSCS subcluster consists of either a pair of nodes or a single node. In the case of single-node MSCS subcluster, the MSCS quorum resource can be configured as a local quorum resource, which means that the quorum resource will be a local disk of that node. This is a preferred configuration since it will save a shared disk per MSCS subcluster.

Some cluster servers, for example, MSCS, have a unique feature in that it remembers the states of resources and resource groups when the cluster was operational. When a node is restarted, MSCS cluster services will bring those resources and resource groups to their previous states. The decisions of bringing resources and resource groups to their on-line and off-line states is made by the multi-cluster service. If an MSCS subcluster (or the node that runs that MSCS subcluster) fails, the cluster service will restart those resources and resource groups that were running on that node on one or more other MSCS subclusters. When the failed node and the corresponding MSCS subcluster is restarted and re-joins the multi-cluster, there will be resource conflicts if the new node and new MSCS subcluster try to bring those resources and resource groups to an on-line state. To resolve this problem, cluster services adds a "hidden" resource into every resource group and makes this hidden resource a dependent resource for all other resources in that resource group. The hidden resource will check the state of its resource group in the multi-cluster configuration database and will fail to start if the resource group is already running on another MSCS subcluster.

The multi-cluster services extend the high availability system clustering features of presently available cluster services to more than two nodes and preserves binary compatibility with presently available cluster services. Referring to FIGS. **3***b* and **3***c*, the present system clustering software **300** consists of two major parts: cluster manager **302** and the cluster services **304**. The cluster manager **302** is designed to manage all resources in a group of clusters **306** and to present a single cluster image to its users. The cluster manager **302** provides an easy-to-use user interface that information technology (IT) administrators are accustomed to. The cluster manager **302** allows administrators to manage a large scale and complex collection of highly available resources in a cluster efficiently and effectively.

The cluster services **304** is a middleware layer that runs on each computer **350** in the cluster. In the depicted example, it comprises a set of executables and libraries that run on the resident Microsoft Windows NT server or other suitable server. The cluster services **304** contains a collection of interacting subsystems. Those subsystems are Topology Services **308**, Group Services **310**, Cluster Coordinator (not shown), CSQL Services **314**, Event Adapters **310**, Recovery Services **316**, and the Cluster API **318**.

The Cluster Coordinator provides facilities for start-up, stop, and restart of cluster services **304**. There is a Cluster Coordinator on each computer in the cluster, but they do not communicate with each other; each one's scope is restricted to the computer on which it runs. The Cluster Coordinator is the component that needs to be started up first. It then brings up the other services in the following order: CSQL Services **314** in stand-alone mode; Topology Services **308**; Group Services **308**; CSQL services **314** in Cluster-mode; Recovery Services **316**; Microsoft Cluster Services (MSCS) Event Adapter; MSCS; and Group Services Event Adapter (GSEA). Further, it monitors each of the other services, and terminates all other services and user applications and restarts the multi-cluster cluster services in case of failures.

Topology Services **308** sends special messages called heartbeats that are used to determine which nodes are active

and running properly. Each node checks the heartbeat of its neighbor. Through knowledge of cluster configuration and alternate communication paths, Topology Services **308** can determine if the loss of a heartbeat represents an adapter failure or a node failure. The MSCS's inter-node heartbeat is ignored in favor of the topology services heartbeat which is multi-cluster wide. Topology Services maintains information about which nodes are reachable from which other nodes, and this information is used to build a reliable messaging facility.

Group Services **310** allows the formation of process groups containing processes on the same or different machines in the cluster. A process can join a group as a provider or a subscriber. Providers participate in protocol actions, discussed in detail below, on the group while subscribers get notified upon changes to the group's state or membership (list of providers). Group Services **310** supports notification on joins and leaves of processes to a process group. It also supports a special, predefined group that a process may subscribe to obtain the status of all cluster nodes. This status is a consistent view of the node status information maintained by Topology Services.

All MSCS subclusters in a multi-cluster are preferably configured as single-node clusters, because Group Services are used for monitoring node up and node down events.

Group Services also provides facilities for cluster-aware applications to handle failure and reintegration scenarios. These facilities are built on top of the reliable messaging facility: Atomic broadcast and n-phase commit protocols for process join, process leave—voluntary and involuntary, process expel, group state change, and provider broadcast messages.

Group Services **310** handles partitioning of the cluster in the following manner. When it recognizes that a cluster that was partitioned has come together, it will generate a dissolve notification to all groups that were part of the partition that has the lesser number of cluster machines. If both partitions have equal number of cluster machines, one of them is chosen to be dissolved.

CSQL Services **314** provides support for a database that can contain configuration and status information. It can function in both stand-alone and cluster modes. Each database is a persistent, distributed resource which, through the use of Group Services **310**, is guaranteed to be coherent and highly available. Each database is replicated across all nodes and checkpointed to disk so that changes are persistent across reboots of the multi-cluster cluster services. CSQL Services **314** ensures that each node has an identical copy of cluster data. CSQL Services also supports transient data that does not persist across reboot but is also consistent on all nodes. Transient data will be initialized to their startup values after a restart of cluster services **304**. CSQL Services **314** supports notification of changes made to the database. Each database can be marked by a three tuple: a timestamp indicating when a database is last modified, ID of the node that proposed the modification, and a CRC checksum. The timestamp is a logical time that is a monotonically increasing number across the entire cluster. CSQL Services **314** runs a Database Conflict Resolution Protocol to determine the most up-to-date replica upon a cluster restart. A node replaces its replica by the cluster's version after making a backup of the existing version of each database when it rejoins a cluster. Modification to a cluster configuration database is permitted only after CSQL transits from stand-alone mode to cluster mode. The conditions for entering cluster mode will be discussed thoroughly below. CSQL Services supports both local and remote client connections.

Event Adapters **312** monitors conditions of subsystems, and generates events when failure situations occur. Events are inserted into a distributed event queue, which is implemented as an event table in the cluster-scope CSQL configuration database. There are four event adapters in a cluster: MSCS Event Adapter that monitors the MSCS subsystem, Group Service Event Adapter that monitors node and network interface failures, Cluster API Event Adapter that converts user requests into multi-cluster events, and a Partition Prevention Event Adapter that monitors for network partitions.

Group Services Event Adapter (GSEA) **310** is a distributed subsystem. Each GSEA instance joins a GSEA Group Services group **311** as a provider. GSEA instances receive LEAVE and FAILURE LEAVE notifications from Group Services and converts them into multi-cluster events. The GSEA group inserts exactly one event into the event queue when a GSEA instance leaves the group either voluntarily or due to failure.

Microsoft Cluster Services Event Adapter (MSCSEA) **320** converts a MSCS notification into events recognizable by the present cluster manager. There is one instance of MSCSEA running on each node. Each MSCSEA monitors MSCS resource groups and MSCS resources that are running on the local node only. Network interface failure and node failure will be detected by the Topology and Group Services subsystem **308**.

Recovery Services **310** is a rule-based, object-oriented, transactional event processing subsystem. Event processing is triggered when a new event is inserted into the cluster-wide event table in a cluster-scope CSQL database. Recovery Services extends the CSQL functionality, adding active and object-oriented SQL statement processing capability to the CSQL subsystem. Methods are expressed in extensions to the SQL language, called active SQL. Specifically, the following statements are introduced: CREATE TRIGGER, EVALUATE, EXECUTE, CONTINUE, CREATE MACRO, and LOAD DLL. The CREATE TRIGGER statement registers a trigger on the specified table with CSQL. When a new row (event) is inserted into the specified table, CSQL will invoke the corresponding event processing rules. Rules are expressed in SQL and the above mentioned active SQL statements. The EVALUATE statement is very similar to SELECT. Instead of selecting a set of data, EVALUATE selects a set of rules and then evaluates those rules. SQL and active SQL statements that are selected and processed by the same EVALUATE statement are part of the same transaction. EXECUTE statements change the physical system state by invoking either a user defined function, an external program, a command file, or a shell script file. CONTINUE statements synchronize event processing among distributed CSQL Servers. In particular, CONTINUE forces synchronization of the CSQL database up to the current point. There may be multiple CONTINUE statements evaluated when an event is processed, forcing multiple database synchronizations. The Create MACRO statement defines a macro, which can be invoked in any SQL statement. A macro returns a data value that can be used in a subsequent SQL statement. LOAD DLL dynamically loads the specified dynamically linked library (DLL) into CSQL. It registers user defined functions in the DLL with CSQL. User defined functions may be invoked either in an EXECUTE statement or embedded in another SQL statement. User defined functions extend the SQL language either by providing commonly used functionality or initiating actions on entities external to the CSQL Server. As an example, user defined functions are used to control MSCS resource management facilities.

Although one embodiment of the cluster services for a multi-cluster is shown, other mechanisms may be used to provide cluster services. For example, the CSQL programming interface uses CSQL language statements. Other types of programming interfaces, data storage, or data registration mechanisms may be employed. In such an implementation, the mechanism would provide consistency of data across the subclusters within the multi cluster, provide consistency of data for the multi-cluster nodes during a reboot, and provide synchronization of data for a new node entering a multi-cluster. In addition, although the recovery services described in the depicted example are an extension of CSQL, such an extension is not necessary in accordance with a preferred embodiment of the present invention.

Multi-cluster API **318** provides access to a multi-cluster as a whole, not a particular MSCS subcluster. It contains functions that operate within the context of the larger cluster but otherwise is functionally identical to the functions of the Microsoft Cluster API. It is intended to be used by Cluster Manager **302** as well as other cluster-aware applications. There is a one-to-one correspondence between functions in the Multi-Cluster API and that of the Microsoft Cluster API. The Multi-Cluster API DLL is binary compatible with the MSCS Cluster API DLL, clusapi.dll. Query Cluster API functions are handled directly by the Multi-Cluster API DLL. Cluster API functions that cause state changes are converted into events which are handled by Recovery Services. Multi-Cluster API DLL uses CSQL Notification mechanisms to wait for the result of event processing. Multi-cluster API DLL communicates with CSQL Services via a well-known virtual IP address. In sum, the cluster services **304** guarantee that the state information put into NT cluster registry by an application program will be available when that application fails over to another node in a cluster. The cluster services **304** provides utilities that examine the system configuration and make sure that it is properly configured. The Cluster Manager **302** will configure, manage, and monitor the multi-clustes and its contained MSCS subclusters. Other utilities may be developed to help simplify the installation process of multiple MSCS subclusters and the multi-cluster cluster services.

The cluster services subsystems are started by the Cluster Coordinator subsystem. The Cluster Coordinator is implemented as an NT service and is started automatically during startup. The Cluster Coordinator then starts all other Cluster Services subsystems in the following order: CSQL Services in stand-alone mode. Topology Services, Group Services, CSQL Services in cluster mode, Recovery Services, MSCS Event Adapter, MSCS, and Group Services Event Adapter.

CSQL Services is initially started in stand-alone mode. Topology Services and Group Services retrieve their configuration information from CSQL databases. After Group Services comes up, CSQL Services forms the CSQL_Services group **315** and runs a Database Conflict Resolution Protocol (DCRP) to synchronize the contents of the cluster configuration database. The first CSQL server forms the group, sets the CSQL_Services group to a BIDDING state, and starts a timer to wait for other CSQL servers to join the group. A CSQL server that joins the group which is in the BIDDING state also starts a timer to wait for others to join. The timer value is defined in the cluster configuration database and may be different from node to node. Inconsistent timer values can be caused by different versions of cluster configuration databases that are being used by different nodes initially. When the first timer expires, the CSQL server broadcasts the timestamp of its cluster configuration database to the group using a Group Services n-phase protocol. Other CSQL servers broadcast their timestamps if their timestamp is more recent than the received one. When multiple CSQL servers send out their timestamp, one will be selected arbitrarily by Group Services and is broadcast to the group in the next phase. A CSQL server sends out its timestamp only if its timestamp is newer than the received timestamp. A CSQL server should send out an older timestamp than the received one only in the first phase to signal other CSQL servers that it has a different version. Eventually the protocol will conclude. Either all CSQL servers have identical timestamps or they all agree on the most up-to-date version. If not all timestamps are identical, the CSQL server that sends out its timestamp last should broadcast its database to all others. CSQL servers should make a backup copy for databases that are to be replaced by the latest version. After CSQL servers synchronize the cluster configuration database, they will set the state of the CSQL_Services group to its RUNNING state. Those CSQL Servers whose replica was replaced by a new version will initiate a restart of Cluster Services. A CSQL server that joins a RUNNING CSQL_Services group must save its replica and replace it by the cluster version regardless of its timestamp value. If the new version has a different timestamp than its existing one which is presently being used by other subsystems, the CSQL Server will initiate a restart of Cluster Services.

The CSQL timestamp is a three tuple: a monotonically increasing number across the entire cluster, the node ID of the node that modified the database the last time, and a CRC check sum.

Once CSQL Services is in RUNNING state, the cluster configuration database, including the event queue is consistent on all nodes. A CSQL server is said to be in cluster mode after it successfully joins a RUNNING CSQL_Services group. Recovery Services, MSCS, MSCS Event Adapter (MSCSEA), and Group Services Event Adapter (GSEA) will then be started. The GSEA joins a GSEA Group Services group and adds a BRING_COMPUTER_UP event for this node into the cluster-wide event queue when processing the Group Services JOIN protocol. Multi-cluster resource groups are initially in an offline state. During the processing of a BRING_COMPUTER_UP event, Recovery Services determines whether any resource group should be brought into an online state.

The DCRP algorithm is summarized below: (1) A CSQL server broadcasts an open database request including the name of the database and a timestamp to the CSQL_Services group, (2) Each CSQL server that has a different timestamp must vote CONTINUE and broadcast its timestamp in the first phase to force a database replication, (3) The CSQL server that receives its own broadcast must vote APPROVE in the first phase, (4) A CSQL server that has an identical timestamp as the received one must vote APPROVE, (5) For each subsequent phase, a CSQL server that has a later timestamp than the received one must broadcast its timestamp and vote CONTINUE, (6) A CSQL server that receives its own timestamp must vote CONTINUE, (7) A CSQL server that has the same or any earlier timestamp must vote APPROVE, (8) If no message was sent in a phase, the server that broadcast its timestamp last must replicate its version of the database to other servers. A server always makes a backup copy of its replica before replacing it.

Still referring to FIGS. 3*b* and 3*c*, the start-up sequence for the multi-cluster system is illustrated. First, the Cluster Coordinator is started as an NT Service during NT startup. The Cluster Coordinator starts and monitors other multi-cluster subsystems. Next, CSQL Services **314** is started in

stand-alone mode. Then, Topology Services **308** is started. Group Services **310** is then started. Next, CSQL Services forms or joins the CSQL_Services group **315**. CSQL Services runs the Database Conflict Resolution Protocol and enters cluster mode Then all cluster scope databases are up-to-date. In particular, the event queue is up to date. Recovery Services **316** is started and the Recovery Services daemon starts both the MSCS Event Adapter **312** and the Group Services Event Adapter (GSEA) **310**, in this order. GSEA forms or joins the GSEA group and it will monitor node failure events. The Recovery Services daemon then inserts a BRING_COMPUTER_UP event for the local node. Recovery Services processes the BRING_COMPUTER_UP event for this node. The MSCS subsystem **306** is started and then monitored by the MSCS Event Adapter **312**. Resource groups are started or moved to this new node depending on resource allocation policy and system status.

Another key feature of the present invention involves a cluster quorum condition. No resource group can be brought into its online state unless one of the following quorum conditions have been met. Cluster Services adopts the same majority quorum scheme that is used in HACMP. Cluster Services uses connectivity information provided by Group Services to Determine majority quorum condition. Additionally, nodes also pass connectivity information through the shared disk path or another method to avoid the split brain problem. When the network is severed and a cluster is divided into several partitions, Cluster services must guarantee not to start a single resource group in multiple partitions at the same time, which can cause corruption of application data on shared disks. The connectivity information passed on the disk path helps each partition to learn about the membership of other partitions and hence help prevent data corruption. A resource group should be brought into an online state on a node within a partition if the following conditions are true: (1) the partition has majority quorum, i.e., more than half of all nodes defined in the cluster configuration database has joined the cluster and is in that partition, or (2) the partition has exactly half of the nodes as defined in the cluster configuration database and no other partitions of the same size exist, or (3) the partition has exactly half of the nodes as defined in the cluster configuration database while another partition contains the other half of the nodes and the node with the lowest ID value is in the former partition.

After starting all Cluster Services subsystems, the Cluster Coordinator will monitor the status of each. If any subsystem terminates abnormally, the Cluster Coordinator will shutdown the node and restart itself, as well as other subsystems. Shutting down a node when any subsystem fails can guarantee that no user applications will continue running when the Cluster Services fails.

When a partition heals, Group Services will dissolve groups in all but one partition. The Group Services daemon on nodes in these "losing" partitions will be terminated. Consequently those nodes will be shut down by the Cluster Coordinator and restarted. The shutdown procedure for Recovery Services must make sure that all resource groups are offline.

Referring to FIG. 3C, the components of support for the present invention are illustrated. Cluster services **304** uses MSCS **306** to manage cluster resources within a node. A resource group is defined in the multi-cluster configuration database first and defined in a MSCS subcluster only if needed. Resource management policy is designed to mimic the MSCS resource management behavior. When a resource group is defined in an MSCS subcluster, the restart flag is always disabled so that a restart decision will be made by event processing subsystem, not by MSCS. A resource group defined in an MSCS subcluster, whether it is a single node cluster, will have at most one node in the preferred node list so that the MSCS auto failover mechanism is disabled. Cluster services will monitor the status of every resource group that is online. When a resource or resource group failure occurs, the MSCS event adapter **312** will insert the corresponding event into the event queue. CSQL services **314** will trigger event processing for the event. One and only one CSQL instance will initiate event processing. Each CSQL instance manages resources including the single-node MSCS subcluster on the local node only. Event processing is designed to be able to handle multiple failures.

Referring to FIGS. **4**, **5**, and **6**, another aspect of the invention involves Event Processing. With respect to FIG. **5**, table **500** illustrates two entries **502** and **504**, which describe two ch_routines: BRING_COMPUTER_UP and NODE—UP. In entry **502**, the action in section **506** corresponds to step **404** in FIG. **4**. In entry **504**, sections **508**, **510**, and **512** contain actions that correspond to steps **408**, **410**, and **414**, respectively. Events defined in Cluster services include, but are not limited to: BRING_COMPUTER_UP, BRING_COMPUTER_DOWN, BRING_RESOURCE_GROUP_ONLINE, BRING_RESOURCE_GROUP_OFFLINE, AND MOVE_RESOURCE_GROUP. When a computer joins a cluster, a "BRING_COMPUTER_UP" event will be inserted into the event queue. To process a BRING_COMPUTER_UP event, the cluster services performs the following: (1) Check whether a quorum exists, and (2) If so, then check whether any resource group should be brought online on the new computer. Some resource groups may be online on some other computer. Those resource groups should be brought into offline state first. Next, the cluster services should bring those resource groups online on the new computer.

All the configuration information, status information, resource management policy, and rules are stored in a cluster scope database, escluster.cfg. Suppose that computer "hilltop" joins a cluster. A BRING_COMPUTER_DOWN event for hilltop is inserted into the event queue, which triggers CSQL to perform event processing wherein a runtime environment is created encapsulates the information relevant to the event and CSQL processes the following statement:

EVALUATE action from ch_routines where ch_routine= "BRING_COMPUTER_UP"

The above statement specifies that statements in the BRING_COMPUTER_UP row of the ch_routines table in the escluster.cfg database should be processed. The actions taken in a ch_routine called BRING_UP_COMPUTER are depicted in table **500** in entry **502**. The ch_resource groups table **600** is defined in FIG. **6**. The table shows one row of the table. Each entry is one column. $_failback_node( ) is a macro which returns a node where the specified resource group should be running based on the specified failback policy and given the fact that a new node rejoins a cluster. $_resource_group_online( ) and $_resource_group_offline( ) are user defined functions that use MSCS Cluster API function calls to bring the specified resource group offline and online on the specified computer node. As a result of processing "EVALUATE action from ch_routines where ch_routine='BRING_COMPUTER_UP'", the following statements are selected and then processed:

"evaluate markup_action from computers where computer=$_get_event_node( );

evaluate action from ch_routines where $_has_quorum( ) and ch_routine=NODE_UP;"

The actions taken for the ch-routine called NODE_UP are illustrated in entry **504** of table **500** in FIG. **5**. As a result of processing the second EVALUATE statement, the following three statements are retrieved and then processed:

evaluate failback_action from ch_resource_groups where current_node<>next_node;

evaluate release_action from ch_resource_groups where current_node <>next_node;

evaluate acquire_action from ch_resource_groups where current_node=" " and next_node=$_get_event_node( ); Those three EVALUATE statements will each search for all ch_resource group rows (object) in the ch_resource_groups table that meets the search condition. When a ch_resource_group row (object) is found, the specified action will be applied to that object. The failback_action contains a single statement, which is:

"update ch_resource_groups set next_node=$_failback_node( ) where ch_resource_group=this ch_resource_group;"

In the above update statement, a macro failback_node( ) is processed which returns a node that is the most preferred node for running the specified resource group given that a new node has just joined the cluster. The update statement stores the returned node name into the next_node column. A macro name is prefixed by $_ to simplify parsing.

The current_node column of a ch_resource_group object indicates the current node where the ch_resource_group is running on. The release_action is processed for this ch_resource_group if the current_node is different from the next node. If that is the case, the following statement is processed;

execute $_resource_group_offline( );

Resource_group_offline( ) is a user defined function which in turn calls the MSCS OfflineResourceGroup( ) function to bring the specified resource group to its offline state. A user defined function is prefixed by $_ to simplify parsing.

Finally, acquire_action is processed on the new node for all the ch_resource_group objects that are not running anywhere and that should be running on the new node. Acquire_action contains one statement:

execute $_resource_group_online( ) resource_group_online( ) is also a user defined function which calls the MSCS OnlineResourceGroup( ) function to bring the specified resource group to its online state.

Cluster Services also supports event simulation. When Recovery Services is invoked to simulate an event, it first clones the cluster configuration database. The event simulation will be performed on the private copy of the configuration database. During a simulation, it is the EXECUTE statement which actually changes the state of physical resources.

FIG. **4** illustrates the method implemented by cluster services when a node wants to join **400** a multi-cluster. First, a node joins the cluster (step **402**). A decision is made as the whether a quorum exists (step **404**). It not, the method returns (step **406**). If a quorum does exist, then for every resource group, the following loop is implemented (step **405**). First a query is made whether any resource group should be failed back to the new node (step **408**). If so, then for each such resource group, the system gets the corresponding MSCS sub-cluster to do an offline of the specified resource group (step **410**). A CONTINUE (step **418**) is performed to synchronize all the nodes. The MSCS sub-cluster on the new node will bring the specified resource group to the online state (step **414**). A query is then made (step **412**) to see if there are more resource groups. If not, the system is done (step **416**); otherwise the method returns to step **405**.

FIG. **4***a* illustrates a flowchart of the method **430** to move a resource group from one node to another. Every node computes the next most preferred node to run the resource group based on node status, the resource group preferred node list, and the failover policy (step **434**). Alternatively, the user can simply specify the next most preferred node. Next, the system queries if the current node is not equal to the next node (step **436**). If not, the system is done (step **438**). If so, then the system gets the MSCS sub-cluster on the current node to bring the specified resource group to offline (step **440**). The process then continues (step **442**). During this step, the system synchronizes its event processing. Afterwards, the system gets the MSCS cluster on the next node to bring the specified resource group to online state (step **444**). Finally, the system is done (step **446**).

FIG. **4***b* illustrates the general method **450** implemented by cluster services when node failure **452** occurs. This method can also be applied to resource failure and resource group failure events. The group service event adapter collectively inserts exactly one node down event into the event queue (step **454**). Node_Down event processing is triggered (step **456**). Next, for every resource group that was running on the failed node, the following steps are applied (step **458**). First, recovery services compute the Next_Node for failover (step **460**). Then a decision is made if My_Node== Next_Node. If not, the system checks if there are more resource groups (step **462**). If so, then the system gets the MSCS sub-cluster to bring the specified resource group online (step **464**). If no more resource groups are available, then the system is done (step **466**). If more are available, then the system loops back to step **458**. While the invention has been described as using MSCS sub-clusters, it is important to understand that this is only one embodiment of the invention. For example, this same system could be built on top of IBM's HACMP or Sun Microsystem's Ultra Enterprise Cluster HA Server to manage these cluster systems. In addition, the apparatus, processes, and instructions of the present invention may be applied to heterogeneous clusters systems. For example, the present invention may be applied to manage a multi-cluster system including a cluster managed using MSCS and a cluster using an Ultra Enterprise Cluster HA server. In addition, the processes of the present invention may be applied to managing multiple processor computers, such as SMP servers.

It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in the form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include recordable-type media such a floppy disc, a hard disk drive, a RAM, and CD-ROMs and transmission-type media such as digital and analog communications links.

The description of the present invention has been presented for purposes of illustration and description, but is not intended to be exhaustive or limited to the invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodi-

ment was chosen and described in order to best explain the principles of the invention, its practical application, and to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use contemplated.

What is claimed is:

1. A method of managing a multi-cluster computer system having at least one node per cluster, said method comprising the steps of:

(a) establishing a multi-cluster computer system comprising at least two subclusters wherein each of the subclusters comprises at least one node;

(b) managing said at least one node with a cluster services program;

(c) failing over between a first node in a first one of the at least two subclusters and any other node within the multi-cluster computer system in response to a failover event; and

(d) returning said system to a consistent state after the failover event.

2. The method of claim 1 wherein said cluster services program manages using a resource API within the at least one node.

3. The method of claim 1 wherein step (a) comprises establishing a multi-cluster of at least three nodes.

4. The method of claim 1 further comprises updating a cluster wide data file.

5. The method of claim 1 wherein step (b) comprises initiating a first cluster services program automatically when said at least one node is booted.

6. The method of claim 5 further comprises initiating a second cluster services program resident on the at least one node after initiating the first cluster services program.

7. The method of claim 6 wherein said first and second cluster services programs are binary compatible.

8. The method of claim 1 wherein step (b) comprises managing a cluster node membership database.

9. The method of claim 1 wherein step (b) comprises managing a heartbeat signal sent between said at least one node and any other node within the multi-cluster computer system.

10. The method of claim 1 wherein step (b) comprises managing inter-node communications between said at least one node and any other node within the multi-cluster computer system.

11. The method of claim 1 further comprises presenting an image of a single cluster with a cluster manager.

12. The method of claim 1 wherein step (b) comprises configuring a multi-cluster quorum resource as a local quorum resource.

13. The method of claim 1 wherein step (d) comprises restarting a node and bringing a shared resource to an initial state.

14. The method of claim 13 wherein step (d) further comprises storing said initial state for said shared resource.

15. The method of claim 1 wherein step (b) comprises, responsive to a conflict for control of a shared resource, failing to restart a failed node which would attempt to control the shared resource.

16. A method of managing a clustered computer system having at least one node, said method comprising the steps of:

(a) establishing a multi-cluster comprising said at least one node and at least one shared resource;

(b) managing said at least one node with a cluster services program;

(c) returning said system to a consistent state after a failover event; and

(d) adding a hidden resource into a resource group on each node.

17. The method of claim 16 further comprises making said hidden resource dependent on any other resource in said resource group.

18. The data processing system of claim 16 wherein (d) comprises means for updating a cluster wide data file.

19. A data processing system for executing a method having a plurality of paths, wherein the data processing system executes native machine code, the data processing system comprising:

(a) means for establishing a multi-cluster computer system comprising at least two subclusters wherein each of the subclusters comprises at least one node;

(b) means for managing said at least one node with a cluster services program;

(c) means for failing over between a first node in a first one of the at least two subclusters and any other node within the multi-cluster computer system in response to a failover event; and

(d) means for returning said system to a consistent state after the failover event.

20. The data processing system of claim 19 wherein said cluster services program manages using a resource API within the at least one node.

21. The data processing system of claim 19 wherein (a) comprises means for establishing a multi-cluster of at least three nodes.

22. The data processing system of claim 19 wherein (b) comprises means for initiating a first cluster services program automatically when said at least one node is booted.

23. The data processing system of claim 22 further comprises means initiating a second cluster services program resident on the at least one node after initiating the first cluster services program.

24. The data processing system of claim 19 wherein (b) comprises means for managing a heartbeat signal sent between said at least one node and any other node within the multi-cluster computer system.

25. The data processing system of claim 19 wherein (b) comprises means for managing inter-node communications between said at least one node and any other node within the multi-cluster computer system.

26. The data processing system of claim 19 further comprises means for presenting an image of a single cluster with a cluster manager.

27. The data processing system of claim 19 wherein (d) comprises means for storing consistent state for a shared resource.

28. The data processing system of claim 19 wherein (b) comprises, responsive to a conflict for control of a shared resource, means for failing to restart a failed node which would attempt to control the shared resource.

29. A data processing system for executing a method having a plurality of paths, wherein the data processing system executes native machine code, the data processing system comprising:

(a) means for establishing a multi-cluster comprising said at least one node and at least one shared resource;

(b) means for managing said at least one node with a cluster services program;

(c) means for returning said system to a consistent state after a failover event; and

(d) means for adding a hidden resource into a resource group on each node.

30. A computer program product in a computer readable medium for executing a method in a data processing system,

(a) first instructions for establishing a multi-cluster computer system comprising at least two subclusters wherein each of the subclusters comprises at least one node;

(b) second instructions for managing said at least one node with a cluster services program;

(c) third instructions for failing over between a first node in a first one of the at least two subclusters and any other node within the multi-cluster computer system in response to a failover event; and

(d) fourth instructions for returning said system to a consistent state after the failover event.

31. The computer program product of claim **30** wherein said second instruction further comprises instruction for managing a cluster node membership database.

32. The computer program product of claim **30** wherein said second instructions further comprises instructions for managing a heartbeat signal sent between said at least one node and any other node within the multi-cluster computer system.

33. The computer program product of claim **30** wherein said second instructions further comprises instructions for managing inter-node communications between said at least one node and any other node within the multi-cluster computer system.

34. A method of managing a clustered computer system having at least one node, said method comprising the steps of:

(a) establishing a multi-cluster comprising said at least one node and at least one shared resource;

(b) managing said at least one node with a cluster services program; wherein said cluster services program manages using a resource API within the at least one node; including managing a heartbeat signal sent between said at least one node and any other node within the multi-cluster;

(c) failing over between a first node and any other node within the multi-cluster;

(d) comprises updating a cluster wide data file;

(e) returning said system to an initial state after a failover event.

35. The method of claim **34** wherein step (a) comprises establishing a multi-cluster of at least two clusters, wherein each of said clusters comprises at least one node.

36. The method of claim **34** wherein step (b) comprises initiating a first cluster services program automatically when said at least one node is booted.

37. The method of claim **34** wherein step (b) comprises managing inter-node communications between said at least one node and any other node within the multi-cluster.

38. The method of claim **34** further comprises presenting an image of a single cluster with a cluster manager.

39. The method of claim **34** wherein step(c) comprises storing said initial state for said shared resource.

40. The method of claim **34** wherein step (b) comprises, responsive to a conflict for control of said shared resource, failing to restart a failed node which would attempt to control said shared resource.

41. A method in a distributed data processing system for managing a plurality of cluster computer systems, the method comprising:

detecting an initiation of a restart of a cluster computer system within the plurality of cluster computer systems, wherein the cluster computer system will restart in a selected state and includes a resource; and

responsive to a determination that the resource is presently operating in another cluster computer system within the plurality of cluster computer systems, preventing a restart of the resource in the cluster computer system.

42. The method of claim **41** wherein the resource is a shared file system.

43. A distributed data processing system, having a plurality of cluster computer systems, comprising:

detection means for detecting an initiation of a restart of a cluster computer system within the plurality of cluster computer systems, wherein the cluster computer system will restart in a selected state and includes a resource; and

preventing means, responsive to a determination that the resource is presently operating in another cluster computer system within the plurality of cluster computer systems, for preventing a restart of the resource in the cluster computer system.

* * * * *

US006347073B1

(12) **United States Patent**
Hiscock et al.

(10) **Patent No.: US 6,347,073 B1**
(45) **Date of Patent: Feb. 12, 2002**

(54) **METHOD AND SYSTEM FOR CONTROLLING DATA TRANSFER BETWEEN A LOGICAL SWITCH SET AND OUTSIDE NODES**

(75) Inventors: **James Scott Hiscock**, Rockport; **Myles Kimmitt**, Shrewsbury, both of MA (US)

(73) Assignee: **3Com Corporation**, Santa Clara, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/069,623**

(22) Filed: **Apr. 29, 1998**

(51) **Int. Cl.**[7] **G01R 31/08**; G06F 11/00; G08C 15/00; H04J 1/16; H04J 3/14

(52) **U.S. Cl.** **370/217**; 370/244; 370/245; 370/248; 714/25

(58) **Field of Search** 370/216, 217, 370/218, 219, 220, 221, 225, 389, 400, 410, 242, 244, 248, 245; 714/1, 25, 26, 27

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,187,706 A * 2/1993 Frankel et al. 370/217
5,737,316 A * 4/1998 Lee 370/248
5,740,157 A * 4/1998 Demiray et al. 370/219
5,909,686 A * 6/1999 Muller et al. 707/104
5,982,743 A * 11/1999 Kusano 370/217
5,982,745 A * 11/1999 Walff et al. 370/219
6,111,852 A * 8/2000 Leung et al. 370/217

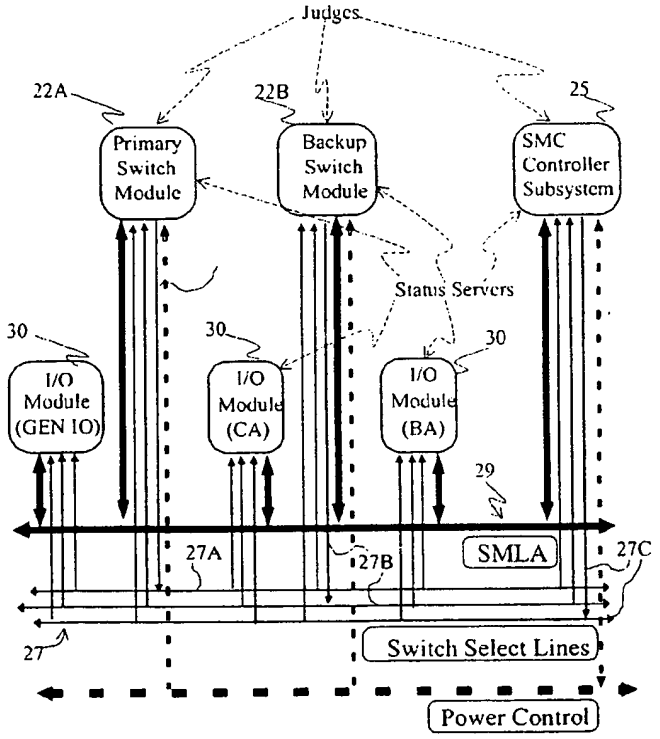
* cited by examiner

Primary Examiner—Ajit Patel
Assistant Examiner—Bob A. Phunkulh

(57) **ABSTRACT**

A plurality of independent control lines are used by I/O modules to determine which switch of a redundant switch set is the active or primary switch. Each line is driven by a different source. Each of these control lines are driven by one of a plurality of judges an each judge can read the other control lines which they are not driving. All the I/O modules can only read the control lines. Each judge makes a decision as to which switch should be the primary switch. Each decision is conveyed using the control lines. The I/O modules use these control lines to direct a multiplexer of the respective outside node to connect to the primary switch. A majority rules algorithm is used to always obtain the correct result in the face of a single error. The criteria used by the judges to decide which switch should be the primary switch is dependent of this mechanism.

17 Claims, 2 Drawing Sheets



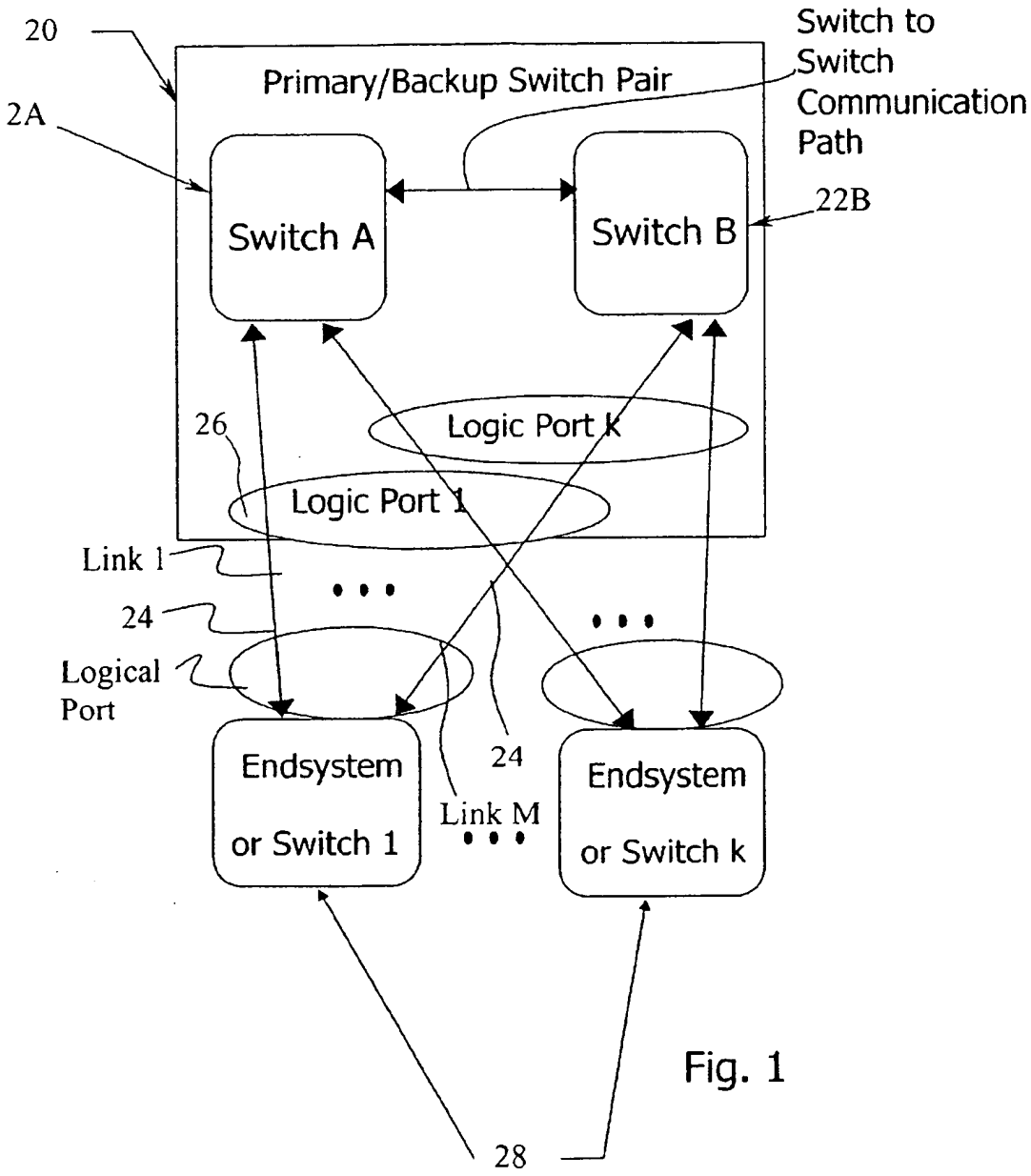

20
2A
Primary/Backup Switch Pair
Switch to Switch Communication Path
Switch A
Switch B
22B
26
Logic Port k
Logic Port 1
Link 1
24
Logical Port
Endsystem or Switch 1
Endsystem or Switch k
24
Link M
28

Fig. 1

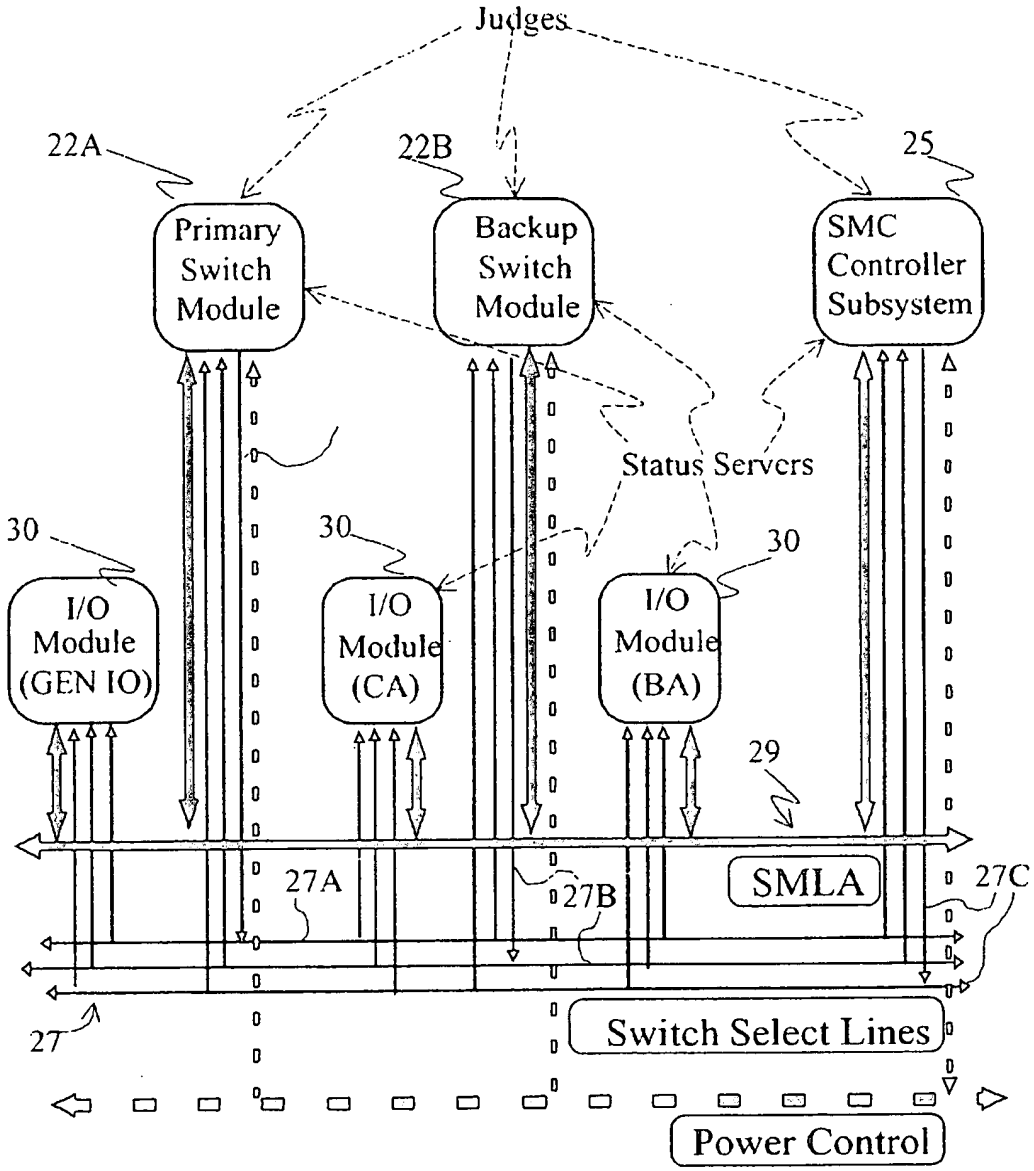

Judges
22A
22B
25
Primary Switch Module
Backup Switch Module
SMC Controller Subsystem
Status Servers
30
30
30
I/O Module (GEN IO)
I/O Module (CA)
I/O Module (BA)
29
SMLA
27A
27B
27C
27
Switch Select Lines
Power Control

Fig. 2

METHOD AND SYSTEM FOR CONTROLLING DATA TRANSFER BETWEEN A LOGICAL SWITCH SET AND OUTSIDE NODES

FIELD OF THE INVENTION

The present invention relates to switches of a computer Local Area Network (LAN), and in particular to controlling the data flow to the plurality of switches inside a Logical Switch Set (LSS).

BACKGROUND OF THE INVENTION

In computer networks, a plurality of end stations can be connected together to form a Local Area Network (LAN). When the number of end stations is small, data can be sent from one end station, to all the other end stations, with each end station only reading data addressed to itself. However, when more and more end stations are connected together, and the amount of data to be transferred between the end stations increases, the amount of time that one station has access to the network decreases. This results in delays in transferring of data from one end station to the other.

Computer network switches, can divide a very large network into a plurality of Local Area Networks. A switch connects one Local Area Network to another Local Area Network, and only transfers data packets from one LAN to another LAN, if that data packet needs to be transferred to the other LAN. In this way, the amount of data packets that are transferred across a network is reduced.

The number of LAN's in computer networks are steadily increasing, and more and more switches are being used to connect the LAN's to each other, as well as to connect previously separate computer networks. This is causing computer networks to evolve into tremendous sizes, with many different possible connections between LAN's and end stations.

Switches are correspondingly transferring a greater number of data packets and the failure of a switch can have large and far ranging consequences. In order to increase the reliability of switches, and to reduce their complexity, Logical Switch Sets (LSS) are formed which are comprised of a plurality of switches. The plurality of switches are grouped together to form the LSS which operates as a single logic packet forwarding device. The grouping together of switches to form an LSS is described in Applicant's co-pending application having Ser. No. 09/014,548 filed on Jan. 28, 1998, and hereby incorporated by reference. The Logical Switch Set (LSS) is connected to a plurality of outside nodes. These outside nodes can be individual LAN's, end stations or ports to other switches, depending on the configuration of the overall network. All of the plurality of outside nodes have connections to each of the switches in the LSS.

In order to increase reliability of the LSS as a whole, one of the switches in the LSS is designated as a primary switch, and the other switches are designated as backup or redundant switches. The outside nodes then transfer the data packets to the primary switch and the primary switch then forwards the data packets to the appropriate output. The operating parameters of the redundant switches are kept similar to the operating parameters of the primary switch, and are updated substantially simultaneously with the primary switch. Should the primary switch fail, or links between the outside nodes and the primary switch fail, the outside nodes then transfer data to one of the backup or redundant switches. The chosen redundant switch then becomes the primary switch.

Coordination and control is needed between the outside nodes and the plurality of switches of the LSS, in order to have all the outside nodes properly recognize one of the switches as a primary switch, and to also recognize a failure of a primary switch, and chose a new primary switch.

SUMMARY AND OBJECTS OF THE INVENTION

It is a primary object of the present invention to provide control and coordination between the outside nodes and the individual switches of an LSS to designate one switch as a primary switch, and to recognize another switch as a primary switch, should the original primary switch fail.

The object of the present invention is accomplished by having each of the switches in the LSS, and a management means in the LSS perform diagnostic tests of the link means that connects the switches to the outside nodes and also tests the respective switch. The results of the diagnostic tests are shared among each of the switches and the management means, as well as the outside nodes, by transmitting the results on to a Switch Management Local Area Network (SMLAN). The SMLAN is separate from, and in parallel with, the link means. In particular each outside node has an I/O means and the SMLAN is connected to the I/O means of each node. A judge means of each switch and management means analyzes the results of the diagnostic tests and generates its own selection signal based on the results. The selection signals from each judge means is sent out on a separate selection line, and each judge means is able to read the selection lines of the other judge means. The I/O modules of the outside nodes are also able to read the selection lines. The selection lines are also separate from, and in parallel with, the link means and SMLAN connecting the outside nodes to the LSS.

Failures in any one of the links of the link means, or in one of the switches, can be difficult to detect, especially by just one switch or judge. A communication error, can be caused by the transmitter, the receiver, or the link connecting the two. A transmitter will not be aware of a communication error unless it receives feedback from the receiver. If a transmitter is supposed to receive feedback, does not receive feedback, the problem can be with the transmitter, the link to the receiver, the receiver, and/or the feedback link from the receiver to the transmitter. Therefore several different diagnostic tests need to be performed, and these tests need to be performed at the transmitter and the receiver, which in the case of the present invention involves both the switches and the I/O modules of the outside nodes. Each switch then makes its own determination as to its abilities to perform properly, and generates a corresponding type of selection signal. The I/O mean of each node reads the selection signals from each of the switches and the management means, and determines which of the switches is to be the primary switch. All subsequent data packets are then transferred to the primary switch. Each of the I/O means uses the same rules for determining the primary switch, and therefore all of the outside nodes simultaneously elect the same switch as the primary switch for transferring the data packets.

The management means provides an independent and unbiased determination to allow a majority rules algorithm to be used and thus always obtain a correct result should a single error occur in somewhere in the network. The management means also acts as a type of "tie-breaker" should two or more switches be equally qualified to perform as the primary switch.

The various features of novelty which characterize the invention are pointed out with particularity in the claims

annexed to and forming a part of this disclosure. For a better understanding of the invention, its operating advantages and specific objects attained by its uses, reference is made to the accompanying drawings and descriptive matter in which a preferred embodiment of the invention is illustrated.

BRIEF DESCRIPTION OF THE DRAWINGS

In the drawings:

FIG. **1** is a schematic diagram showing the general scheme of the logical switch set according to the invention;

FIG. **2** is a schematic view showing the logical switch set of FIG. **1** as a primary/backup pair implemented with a controller for primary switch selection.

DESCRIPTION OF THE PREFERRED EMBODIMENT

Referring to the drawings, and in particular to FIG. **1**, the invention comprises a Logical Switch Set (LSS) **20** which comprises two or more switches **22** that together act as a single packet forwarding device with specific connection rules. The LSS is a Redundant Switch Set (RSS), providing a primary/backup switch pair **22***a* and **22***b*. The switches **22***a* and **22***b* act as one logical device and have one logical port **26** for each outside node **28** which can either be an end system or an outside switch. Links **24** are provided for connecting the outside nodes **28** to the individual switches **22**. All of the links **24** combined to form the link means connecting the outside nodes to the LSS **20**. The outside nodes **28** are multiplexed to the switches **22** of the LSS **20** through the links **24**. One switch assumes the role of the primary switch, providing the LAN interconnection and management, while the other switch is the backup switch, and stands ready to take over in the event of a failure on the primary switch. Only one link is active at a time, therefore only one of the two switches is supplying traffic to the LSS. Implementation techniques vary with network technology, and implementations are available for Ethernet, fast Ethernet, FDDI, token ring and ATM (asynchronous transfer node). In the preferred embodiment, there are two switches **22***a* and **22***b*. Each of the outside nodes **28** has an I/O module or means **30** which are connected to the switches **22** by the SMLAN **29** and the switch select lines **27**. The LSS includes a management means **25** in the form of a switch management controller (SMC) and controller subsystem. The management means allows a majority rules algorithm to be used in the determination process and thus always obtain a correct result should a single error occur in somewhere in the network. The management means also provides an independent and unbiased determination to act as an impartial tie-breaker in the case of two or more switches being equally qualified to be a primary switch. Each outside node **28** has its own logical port **26** on the LSS **20**. In this way, the plurality of switches **22** acts as a single logical switch and therefore can be easily integrated into the topology of the computer network.

Each of the switches **22**, and the management means **25** include judge means for detecting a status of the link means and the respective switch **22**. The judge means performs diagnostic tests and the results of these tests are shared among the other judges and I/O modules over the SMLAN **29**. The SMLAN **29** and the switch select lines **27** are out of band signals or lines that are separate from, but parallel with, the links **24** of the link means.

The judges and I/O means **30** together form status servers which register, implicitly or explicitly, the ability to test the packet switch fabric and report the results on the SMLAN **29**. The judges discover and coordinate the servers to gather switch fabric status. The judges gather information about the packet switch fabric status and each judge expresses their belief as to which switch should be active on their own respective switch select line **27**. The I/O means **30** receive the active switch judgements from all three judges via the switch select lines **27** and determine from which switch module to receive traffic based on the voting of the three judges. This provides a voting system for an effective switch selection. Each judge can read the other two lines they are not driving. All the I/O means **30** can only read the switch select lines. A majority rules algorithm is used to always obtain the correct results in the face of a single error. The criteria used by the judges to decide which switch should be the primary switch is dependent of this mechanism.

Some of the criteria that can be used, is for every switch, and management means to send a "hello" signal to everybody else over the SMLAN **29**. The judges would then use the present or absence of a "hello" signal to determine if a switch and it corresponding links are capable of being a primary switch. Another criteria for a backup switch to become a primary switch is if the backup switch has a set of ports which is a superset of the ports presently being served by the primary switch. A more involved criteria would have each node sending and confirming test messages through the switches.

In an embodiment with two switches **22***a*, **22***b* in an LSS **20**, there will be a management means **25**, to create a total of three switch select lines **27***a* corresponding to switch **22***a*, switch select line **27***b* corresponding to switch **22***b* and switch select line **27***c* corresponding to management means **25**. The switches **22***a*, **22***b* transmit a 0 signal onto their respective switch select lines if they feel that they are qualified to be the primary switch. The switches **22***a*, **22***b* transmit a 1 signal onto their respective switch select lines if they feel that they are not qualified to be the primary switch. The management means **25** transmits a 0 onto switch select line **27***c* if it determines that switch **22***a* should be the primary switch, and transmits a 1 onto switch select line **27***c* if it determines that switch **22***b* should be the primary switch, Each of the I/O modules reads the switch select lines **27***a–c*, and determines the primary switch module according to the following table.

Select line 27A	Select line 27B	Select line 27C	Result
0	0	0	Switch 22A
0	0	1	Switch 22B
0	1	0	Switch 22A (no error)
0	1	1	Switch 22A
1	0	0	Switch 22B
1	0	1	Switch 22B (no error)
1	1	0	Switch 22A
1	1	1	Switch 22B

If the reading of the switch select lines **27**A–C is 000 or 010 or 011 or 110, switch **22**A is enabled as the primary switch. If the switch select lines **27**A–C read 001 or 100 or 101 or 111, the switch **22**B is enabled as the primary switch.

The type of diagnostic tests that can be performed in an LSS **10** are further described in Applicant's co-pending application 09/014,548.

The judges and the I/O means thus form a voting and vote collection system that will always uniformly determine a capable one of the switches and corresponding link to which

the outside nodes can transmit the data. The voting and vote collection system uniformly determine the capable switch both when the multiplexer control system is free of errors and includes a single error.

The present invention is not limited to the above embodiment, but instead can be used with any number of switches, or the I/O means **30** could use a different algorithm based on the selection signals read from the switch select lines.

The features described in the specification, drawings, abstract and claims can be used individually, and in arbitrary combinations, for practicing the present invention.

While specific embodiments of the invention have been shown and described in detail to illustrate the application of the principles of the invention, it will be understood that the invention may be embodied otherwise without departing from such principles.

What is claimed is:

1. A multiplexer control system for an outside node of are redundant computer network switch set, the system comprising:

link means connecting said switch set to the outside node;

two switches grouped together to form the switch set as a single logic packet forwarding device, one of said switches functioning as a primary switch, and a remainder of said switches being a backup switch, said link means including a separate link from each of said switches to the outside node, said each switch including judge means for detecting a status of said link means, each said judge means generating a selection signal based on a respective detected status of said link means, said each judge means of said switches performs diagnostic tests of a respective switch and said link means;

management means connected to said two switches and for also detecting a respective status of said link means and said switches, said management means also including judge mean for generating a separate selection signal based on said respective detected status of said switches and said link means, said judge means of said management means performs diagnostic tests of said switch means and said link means, a type of said selection signal is based on results from respective said diagnostic tests, said each judge means of said two switches generates a first signal if said results indicate said respective switch is operational, said each judge means of said two switches generates a second signal if said results indicate said respective switch is non-operational, said judge means of said management means generates said first signal if said results indicate a first of said two switches is better qualified to receive said data from the outside node, said judge means of said management means generates said second signal if said results indicate a second of said two switches is better qualified to receive said data from the outside node;

a plurality of switch select lines for transmitting said separate selection signals from said switches and said management means to the outside node;

means connected to the outside node and reading said selection signals, said I/O means determining to which of said links and said switches the outside node will transmit data based on said selection signals.

2. A method for controlling data flow to a logical switch set, the process comprising:

detecting a status of each switch in said logical switch set and corresponding link of the logical switch set;

generating a selection signal from each said switch based on said respective detected status of said links;

generating a second selection signal from a management means ofthe logical switch set, said management means being connected to said plurality of switches and detecting a status of said links and said switches, said selection signal generated by said management means being based on said respective detected status of said switches and said links;

transmitting said selection signals to a plurality of outside nodes;

determining in each of said outside nodes to which of said links and said switches the outside node will transmit data based on said selection signals, said determining including forming a voting and vote collection system that will always uniformly determine a capable one of said switches and corresponding link when the logical switch set is free of errors and includes a single error.

3. A method in accordance with claim **2**, wherein:

said selection signals are transmitted separately and in parallel with said data.

4. A method in accordance with claim **2**, wherein:

said detecting of status and generating of respective said selection signals is performed separately by each said switch and management means separately performing diagnostic tests.

5. A method in accordance with claim **2**, wherein:

said detecting includes performing diagnostic tests by each of said switches and management means;

a type of said selection signal generated is determined by results of said diagnostic tests.

6. A method in accordance with claim **5**, further comprising:

transmitting said results of said diagnostic tests of each said switch and management means to all other said switches, said management means and said outside nodes;

generating said type of selection signal at each said switch based on received said results of said diagnostic tests of all of said switches and said management means.

7. A method in accordance with claim **6**, further comprising:

transmitting said selection signal of each said switch and management means to all other said switches, said management means and said outside nodes;

generating said type of selection signal at each said switch based on received said selection signals.

8. A method in accordance with claim **2**, wherein:

said selection signal is generated by each said switch to determine if a respective said switch is qualified to transfer data between the outside nodes.

9. A multiplexer control system for an outside node of a redundant computer network switch set, the system comprising:

link means connecting said switch set to the outside node;

a plurality of switches grouped together to form the switch set as a single logic packet forwarding device, one of said switches functioning as a primary switch, and a remainder of said switches being backup switches, said link means including a separate link from each of said switches to the outside node, said each switch including judge means for detecting a status of said link means, each said judge means generating a second selection signal based on a respective detected status of said link means, said each judge

means performs diagnostic tests of a respective switch and said link means, a type of said selection signal is based on results from respective said diagnostic tests;

management means connected to said plurality of switches and for also detecting a respective status of said link means and said switches, said management means also including judge means for generating a second selection signal based on said respective detected status of said switches and said link means;

a plurality of switch select lines for transmitting said separate selection signals from said switches and said management means to the outside node;

I/O means connected to the outside node and reading said selection signals, said I/O means determining to which of said links and said switches the outside node will transmit data based on said selection signals, said judge means and said I/O means form a voting and vote collection system that will always uniformly determines a capable one of said switches and corresponding link when the multiplexer control system is free of errors and includes a single error.

10. A system in accordance with claim **9**, wherein:

said each judge means of said switches performs diagnostic tests of a respective switch and said link means, said judge means of said management means performs diagnostic tests of said switch means and said link means, a type of said selection signal is based on results from respective said diagnostic tests.

11. A system in accordance with claim **9**, wherein:

said each judge means reads said selection signals from remaining said judge means;

said type of selection signal generated by said respective judge means being also based on said selection signals received from said remaining means.

12. A system in accordance with claim **9**, wherein:

said I/O means uses a majority rules algorithm to determine to which of said links and said switches the outside node will transmit said data.

13. A system in accordance with claim **9**, further comprising:

a plurality of the outside nodes connected to said each switch by said link means, each outside node reading said selection signals and determining to which of said links and said switches each outside node will transmit data.

14. A system in accordance with claim **13**, wherein:

each outside node similarly determines to which of said links and said switches each outside node will transmit data.

15. A system in accordance with claim **9**, wherein:

said switch select lines are separate from, and are in parallel with, said link means transferring data from said the outside node to the logical switch set.

16. An apparatus in accordance with claim **9**, wherein:

said each judge means of said switches generates said selection signals to indicate if a respective said switch is qualified to transfer data between the outside nodes.

17. A multiplexer control system for an outside node of a redundant computer network switch set, the system comprising:

link means connecting said switch set to the outside node;

a plurality of switches grouped together to form the switch set as a single logic packet forwarding device, one of said switches functioning as a primary switch, and a remainder of said switches being backup switches, said link means including a separate link from each of said switches to the outside node, said each switch including judge means for detecting a status of said link means, each said judge means generating a second selection signal based on a respective detected status of said link means, said each judge means performs diagnostic tests of a respective switch and said link means, a type of said selection signal is based on results from respective said diagnostic tests, said each judge means of said switches generates a first signal if said results indicate said respective switch is operational, said each judge means of said switches generates a second signal if said results indicate said respective switch is non-operational;

management means connected to said plurality of switches and for also detecting a respective status of said link means and said switches, said management means also including judge means for generating a second selection signal based on said respective detected status of said switches and said link means;

a plurality of switch select lines for transmitting said separate selection signals from said switches and said management means to the outside node;

I/O means connected to the outside node and reading said selection signals, said I/O means determining to which of said links and said switches the outside node will transmit data based on said selection signals.

* * * * *

US006173411B1

(12) **United States Patent**
Hirst et al.

(10) **Patent No.: US 6,173,411 B1**
(45) **Date of Patent: Jan. 9, 2001**

(54) **METHOD AND SYSTEM FOR FAULT-TOLERANT NETWORK CONNECTION SWITCHOVER**

(75) Inventors: **Michael D. Hirst**, Lakeville; **Alan A. Gale**, Carver; **Gene A. Cummings**, Sherborn, all of MA (US)

(73) Assignee: **The Foxboro Company**, Foxboro, MA (US)

(*) Notice: Under 35 U.S.C. 154(b), the term of this patent shall be extended for 0 days.

(21) Appl. No.: **09/059,896**

(22) Filed: **Apr. 14, 1998**

Related U.S. Application Data

(60) Provisional application No. 60/062,681, filed on Oct. 22, 1997, and provisional application No. 60/062,984, filed on Oct. 21, 1997.

(51) **Int. Cl.**[7] **G06F 11/00**
(52) **U.S. Cl.** **714/4**; 709/223
(58) **Field of Search** 714/2, 3, 4, 7, 714/18, 47, 51; 709/223, 238, 239, 240, 241, 242

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,692,918	*	9/1987	Elliott et al.	370/85
4,710,926		12/1987	Brown et al.	371/9
4,787,082		11/1988	Delaney et al.	370/85
4,964,120		10/1990	Mostashari	370/16
5,153,874		10/1992	Kohno	370/13
5,159,685	*	10/1992	Kung	714/712
5,218,600		6/1993	Schenkyr et al.	370/16
5,276,440		1/1994	Jolissaint et al.	340/825.02
5,329,521		7/1994	Walsh et al.	370/16
5,337,320	*	8/1994	Kung	714/712
5,341,496		8/1994	Middledorp et al.	395/575
5,390,326		2/1995	Shah	395/575
5,485,465		1/1996	Liu et al.	395/182.02
5,485,576		1/1996	Fee et al.	395/185.09
5,493,650		2/1996	Reinke et al.	395/200.12
5,508,997		4/1996	Katou	370/16
5,508,998		4/1996	Sha et al.	370/16.1
5,586,112		12/1996	Tabata	370/221
5,661,719		8/1997	Townsend et al.	370/216
5,675,723		10/1997	Ekrot et al.	395/182.02
5,680,437		10/1997	Segal	379/10
5,987,521	*	11/1999	Arrowood et al.	709/239

OTHER PUBLICATIONS

Stevens, et al. "TCP/IP Illustrated, vol. 1. The Protocols," *TCP/IP Illustrated* vol. 1, XP-002106390, pp. 85–96.

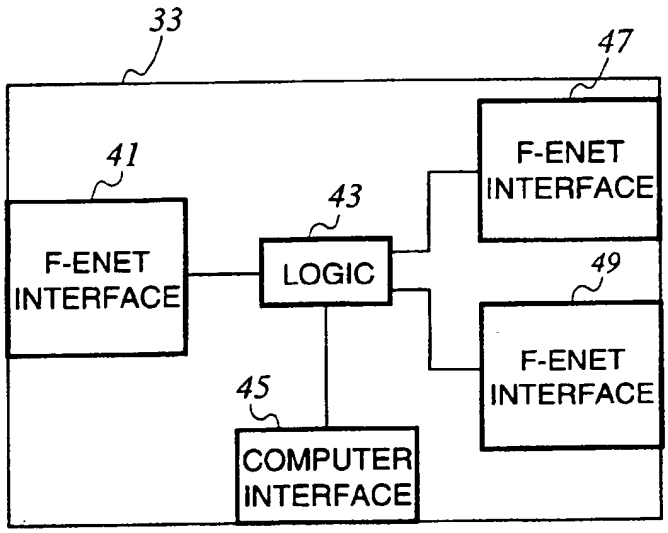
* cited by examiner

Primary Examiner—Robert W. Beausoliel, Jr.
Assistant Examiner—Pierre Eddy Elisca
(74) *Attorney, Agent, or Firm*—David Barron; Sean Detweiler; J. J. Morris

(57) **ABSTRACT**

A computer is connected to redundant network switches by primary and secondary connections, respectively. Test messages are sent across each connection to the attached switches. A break in a connection, or a faulty connection, is detected upon a failed response to one of the test messages. In response to this failure, traffic is routed across the remaining good connection. To facilitate fast protocol rerouting, a test message is sent across the now active connection bound for the switch connected to the failed connection. This message therefor traverses both switches causing each to learn the new routing. Rerouting is therefor accomplished quickly.

19 Claims, 9 Drawing Sheets



PRIOR ART

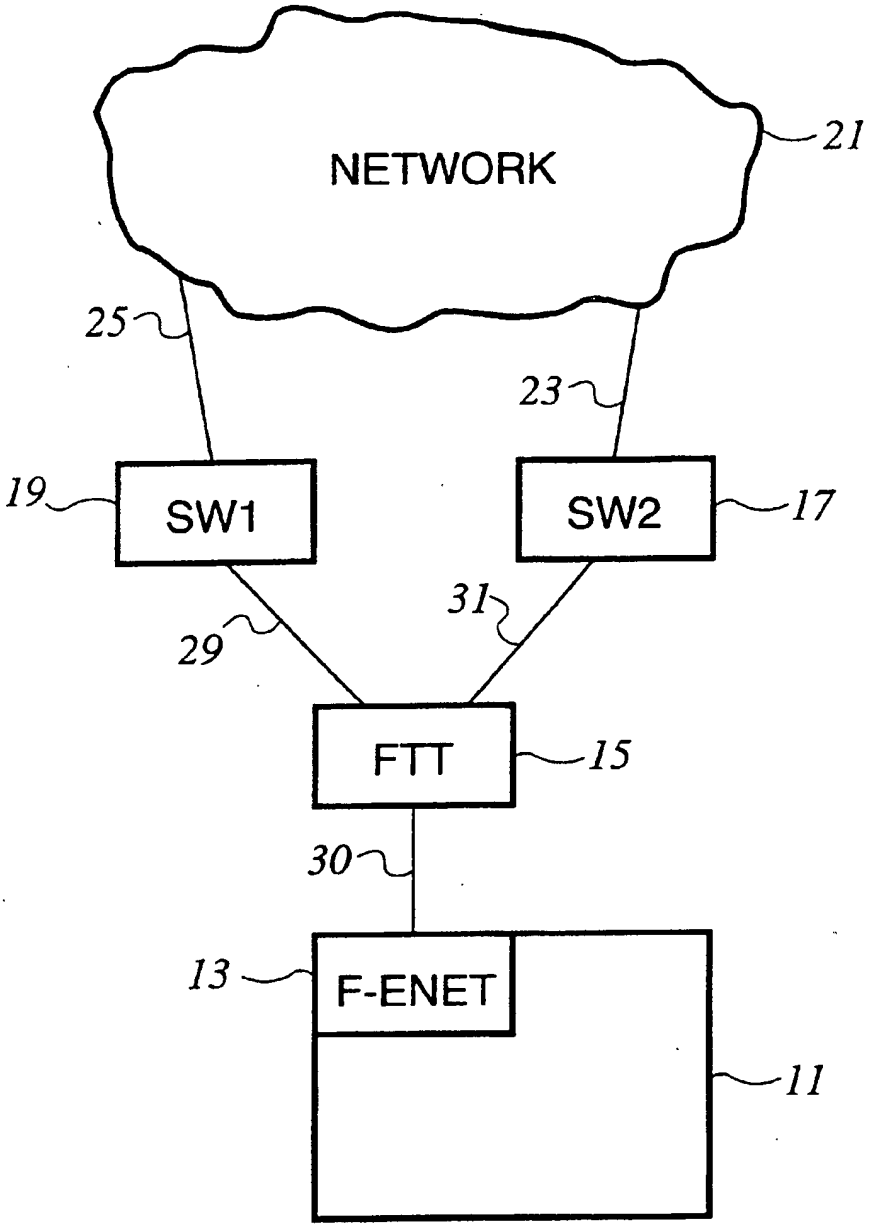


FIG. 1

PRIOR ART

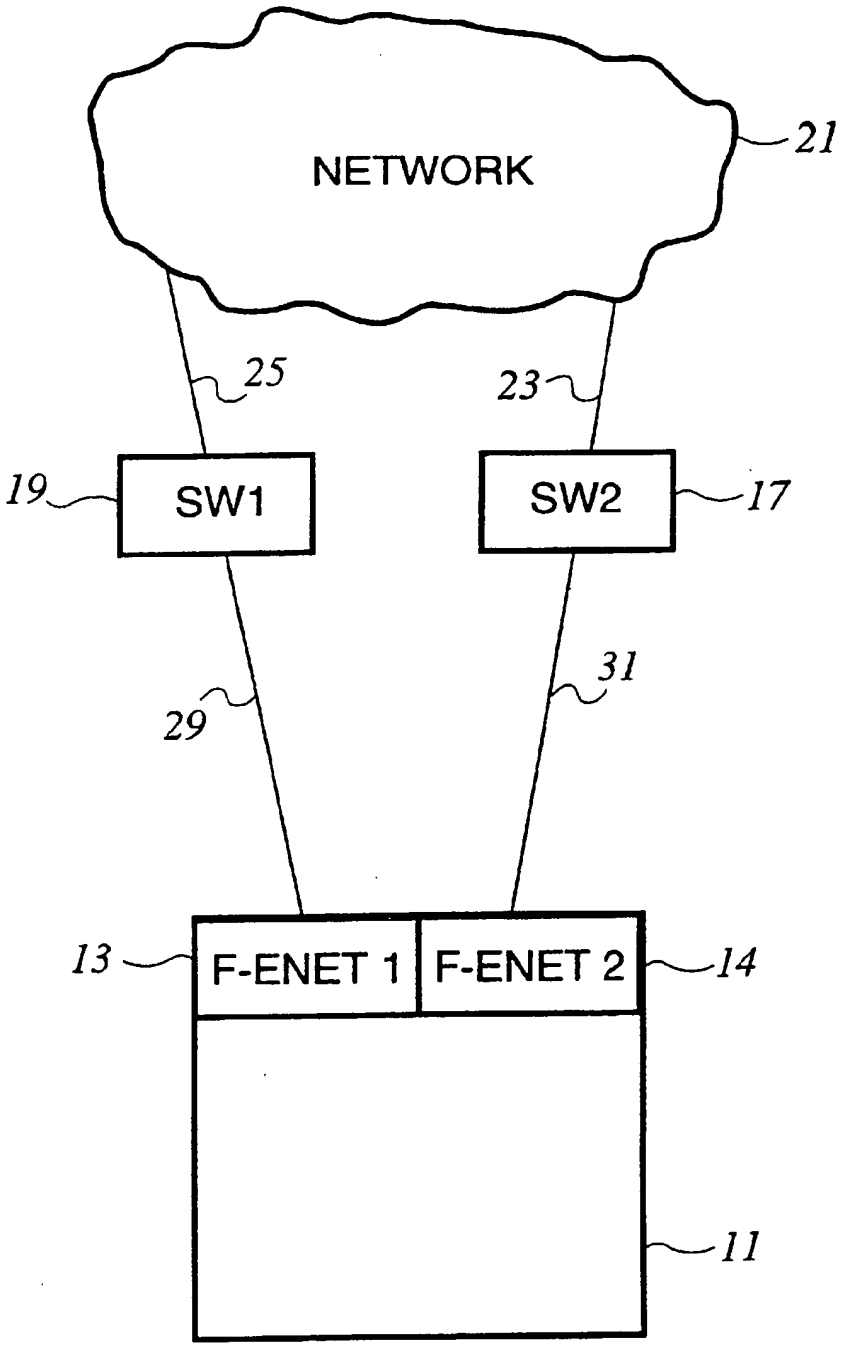


FIG. 2

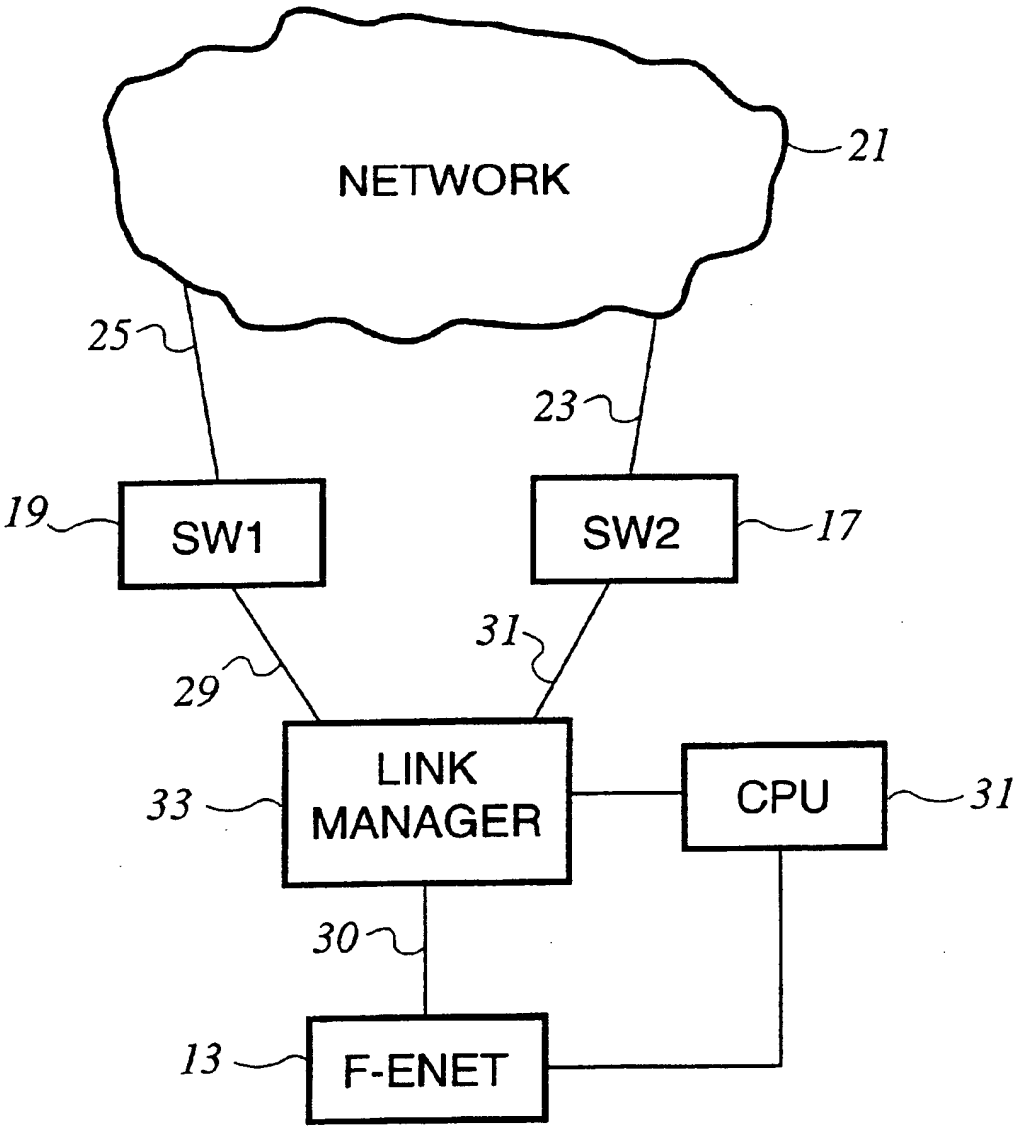

NETWORK
21
25
23
19
SW1
SW2
17
31
29
LINK
MANAGER
33
CPU
31
30
13
F-ENET

FIG. 3

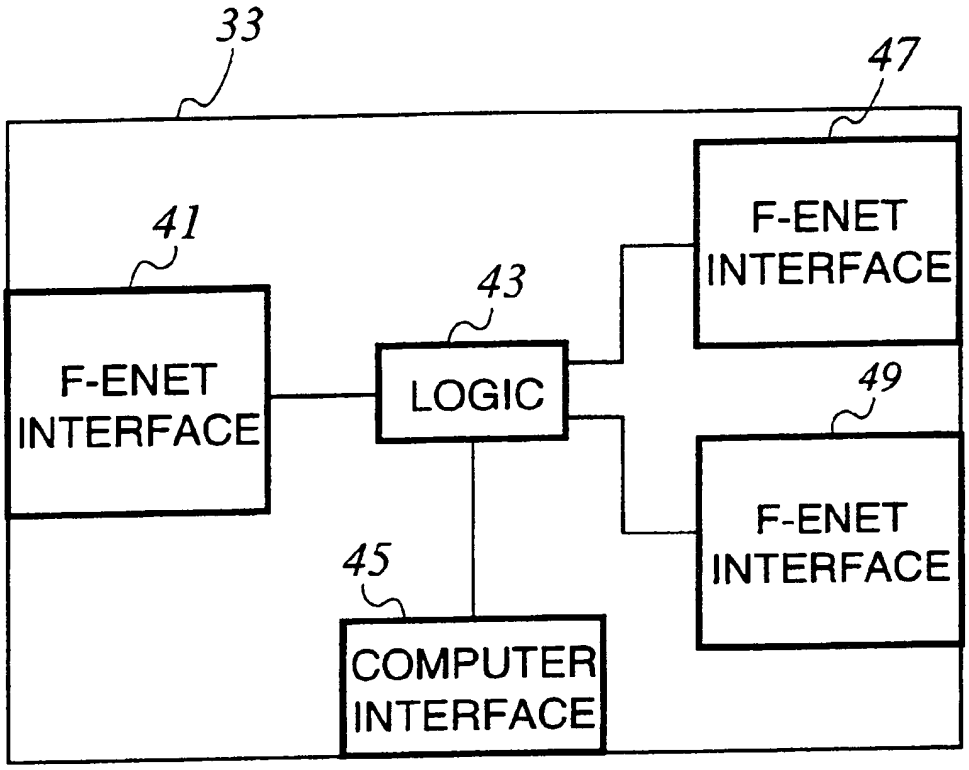

33
47
41
F-ENET INTERFACE
43
LOGIC
F-ENET INTERFACE
49
F-ENET INTERFACE
45
COMPUTER INTERFACE

FIG. 4

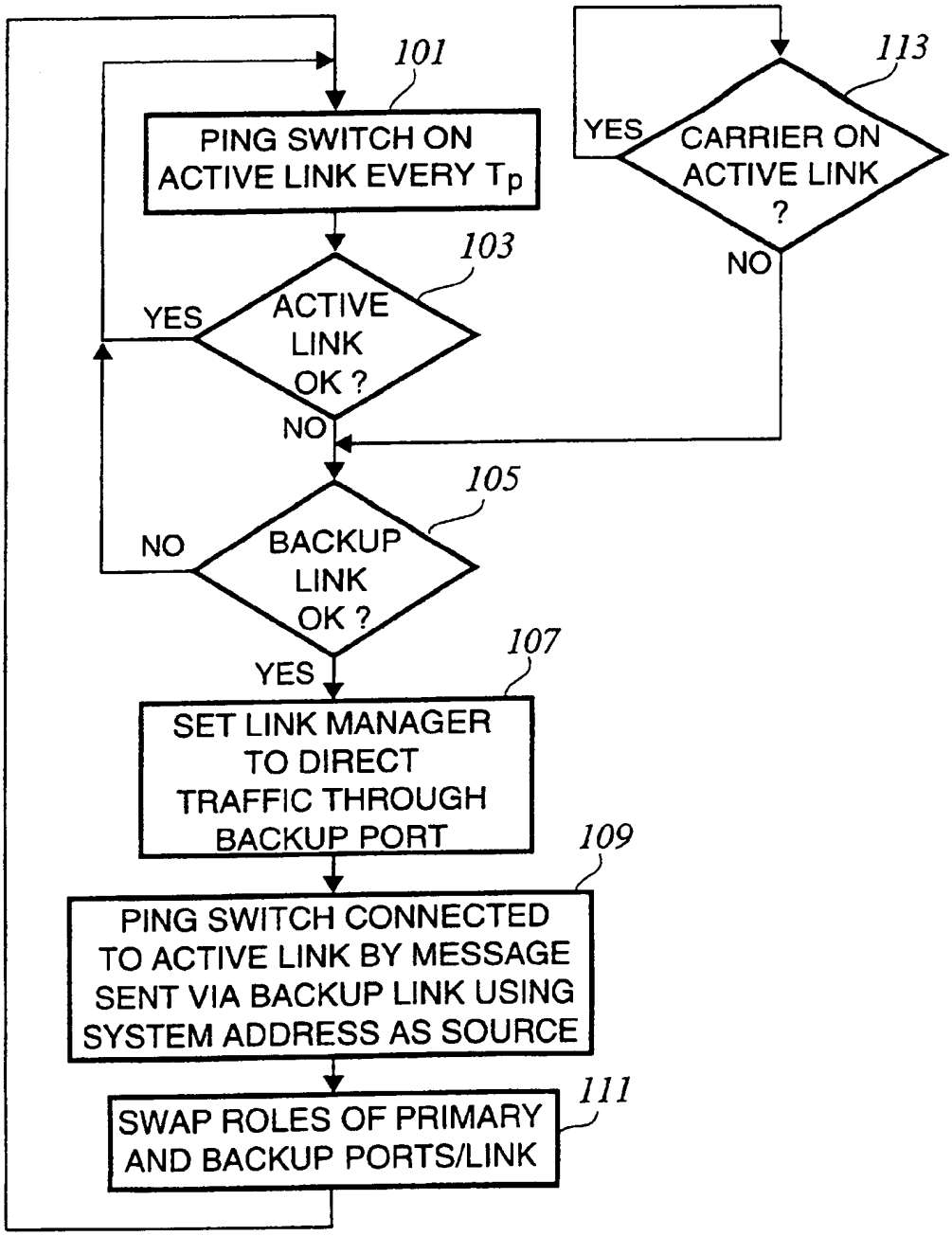

101
PING SWITCH ON ACTIVE LINK EVERY Tp
103
ACTIVE LINK OK ?
YES
NO
105
BACKUP LINK OK ?
NO
YES
107
SET LINK MANAGER TO DIRECT TRAFFIC THROUGH BACKUP PORT
109
PING SWITCH CONNECTED TO ACTIVE LINK BY MESSAGE SENT VIA BACKUP LINK USING SYSTEM ADDRESS AS SOURCE
111
SWAP ROLES OF PRIMARY AND BACKUP PORTS/LINK
113
CARRIER ON ACTIVE LINK ?
YES
NO

FIG. 5

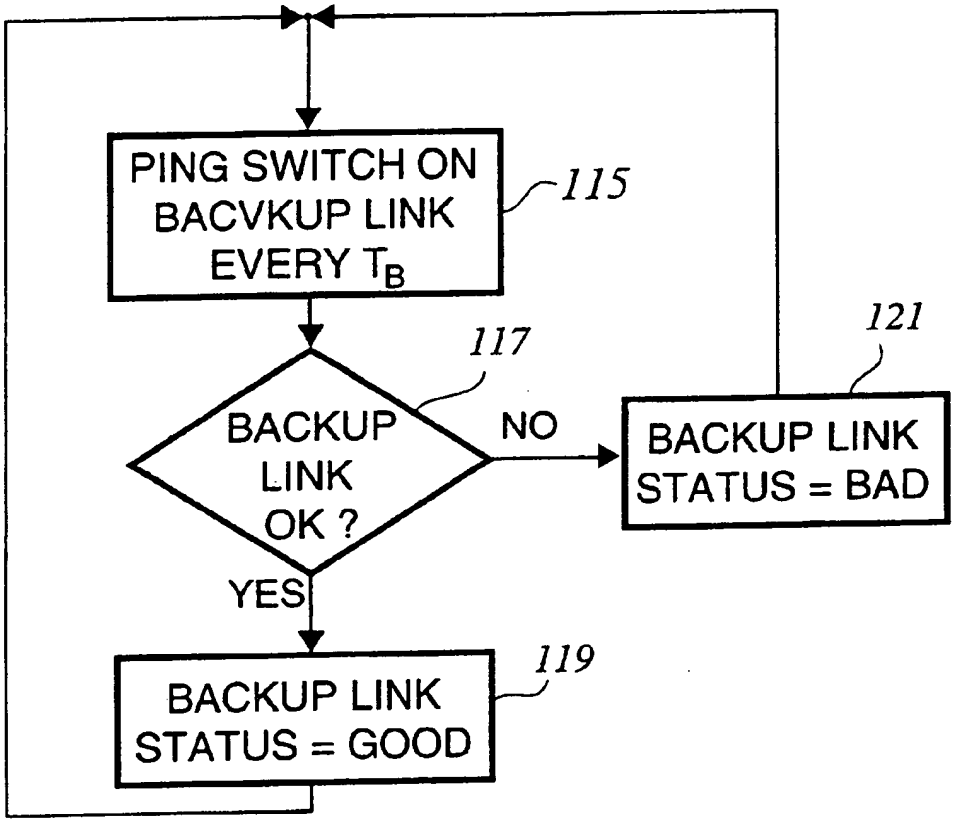

PING SWITCH ON BACVKUP LINK EVERY T_B
115
117
BACKUP LINK OK ?
NO
121
BACKUP LINK STATUS = BAD
YES
119
BACKUP LINK STATUS = GOOD

FIG. 6

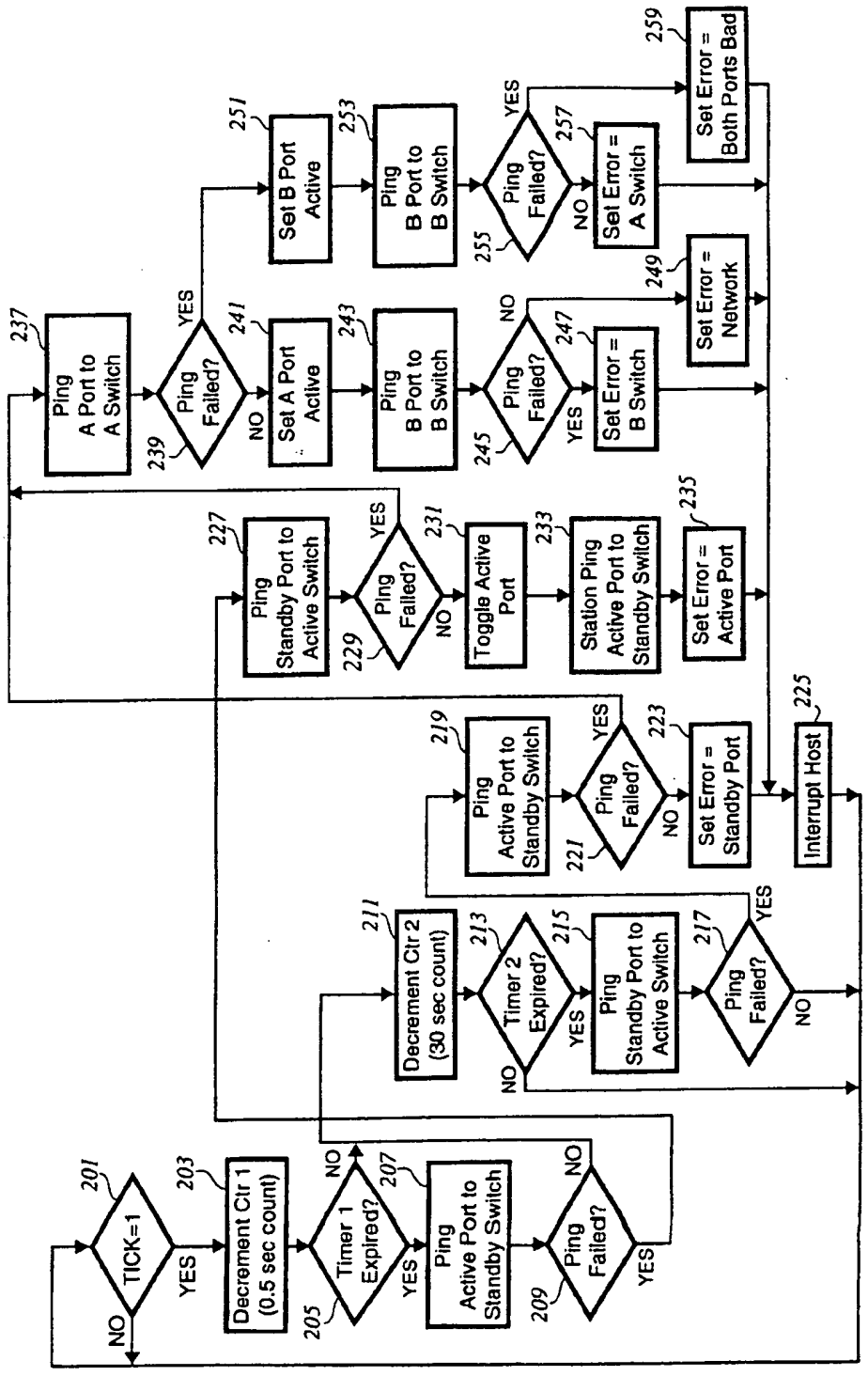

201
TICK=1
NO
YES
203
Decrement Ctr 1 (0.5 sec count)
205
Timer 1 Expired?
NO
YES
207
Ping Active Port to Standby Switch
209
Ping Failed?
NO
YES
211
Decrement Ctr 2 (30 sec count)
213
Timer 2 Expired?
NO
YES
215
Ping Standby Port to Active Switch
217
Ping Failed?
NO
YES
219
Ping Active Port to Standby Switch
221
Ping Failed?
NO
YES
223
Set Error = Standby Port
225
Interrupt Host
227
Ping Standby Port to Active Switch
229
Ping Failed?
NO
YES
231
Toggle Active Port
233
Station Ping Active Port to Standby Switch
235
Set Error = Active Port
237
Ping A Port to A Switch
239
Ping Failed?
NO
YES
241
Set A Port Active
243
Ping B Port to B Switch
245
Ping Failed?
NO
YES
247
Set Error = B Switch
249
Set Error = Network
251
Set B Port Active
253
Ping B Port to B Switch
255
Ping Failed?
NO
YES
257
Set Error = A Switch
259
Set Error = Both Ports Bad

FIG. 7

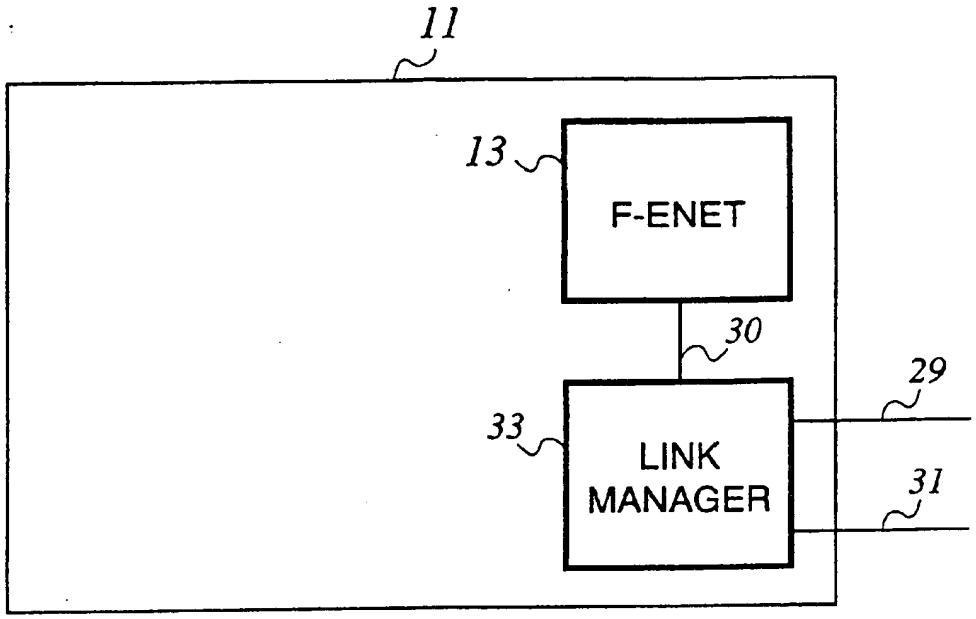

11
13
F-ENET
30
29
33
LINK
MANAGER
31

FIG. 8

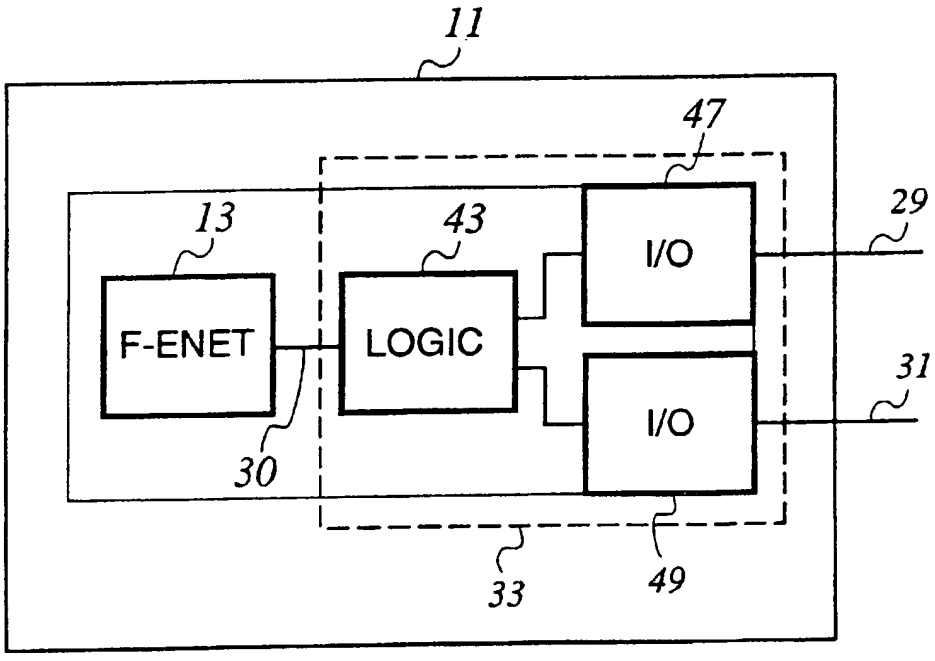

11
47
13
43
I/O
29
F-ENET
LOGIC
31
I/O
30
33
49

FIG. 9

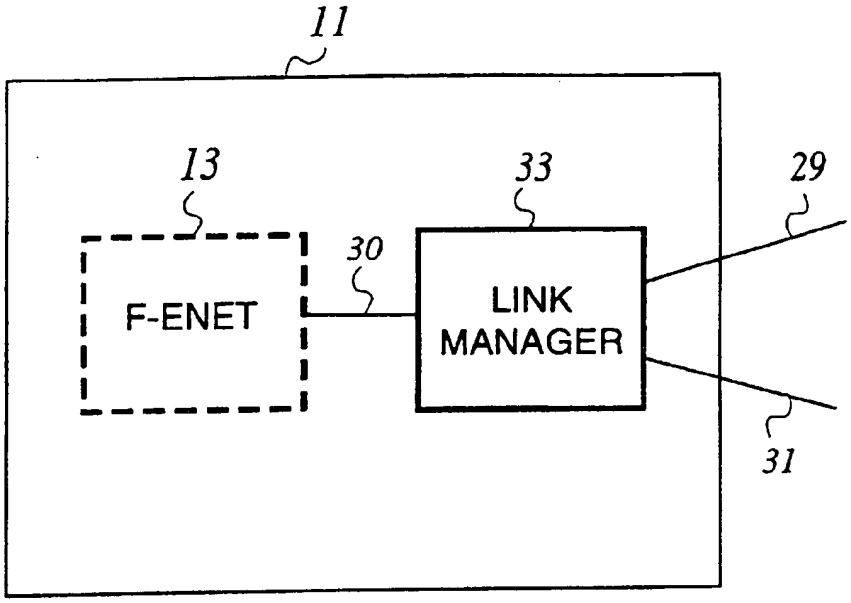

11
13
33
29
30
F-ENET
LINK
MANAGER
31

FIG. 10

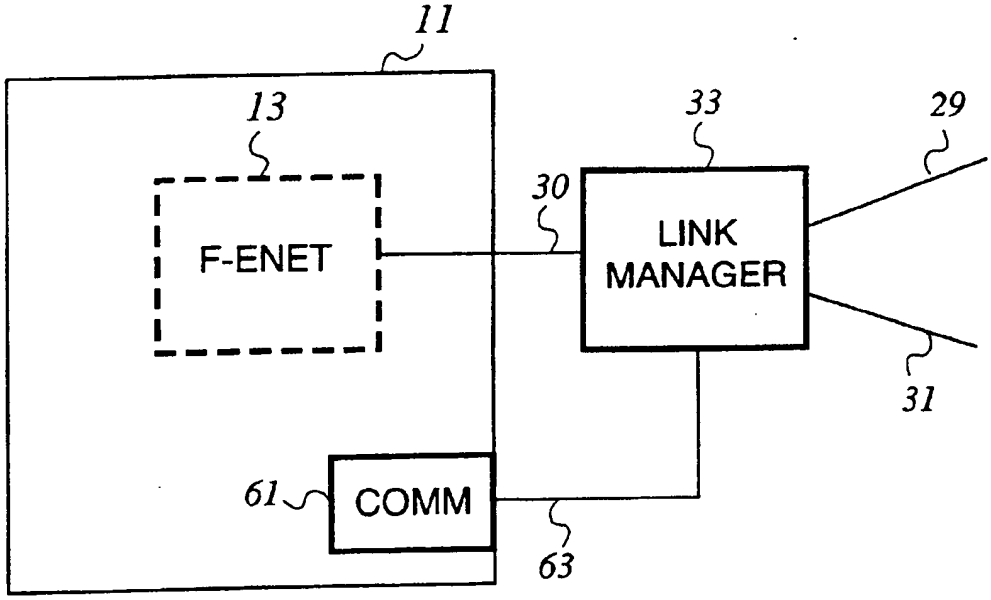

11
13
33
29
30
F-ENET
LINK
MANAGER
31
61
COMM
63

FIG. 11

METHOD AND SYSTEM FOR FAULT-TOLERANT NETWORK CONNECTION SWITCHOVER

CROSS REFERENCING TO RELATED PATENTS

This patent application is related to co-pending patent applications: "Fast Re-Mapping For Fault Tolerant Connections" Ser. No. 60/062,681, Filed: Oct. 20, 1997; and "Fast Re-Mapping For Fault Tolerant Connections", Ser. No. 60/062,984, Filed: Oct. 21, 1997 both of which are incorporated by reference herein in their entireties.

TECHNICAL FIELD

The present invention relates, in general, to fault-tolerant computing. More specifically, the present invention relates to methods and systems for quickly switching between network connections.

BACKGROUND OF THE INVENTION

The reliability of computer based applications continues to be an important consideration. Moreover, the distribution of applications across multiple computers, connected by a network, only complicates overall system reliability issues. One critical concern is the reliability of the network connecting the multiple computers. Accordingly, fault-tolerant networks have emerged as a solution to insure computer connection reliability.

In many applications, the connection between a single computer and a network is a critical point of failure. That is, often a computer is connected to a network by a single physical connection. Thus, if that connection were to break, all connectivity to and from the particular computer would be lost. Multiple connections from a single computer to a network have therefore been implemented, but not without problems.

Turning to FIG. 1, a diagram of a computer **11** connected to a network **21** is shown. Computer **11** includes a network interface, for example, a fast-Ethernet interface **13**. A connection **30** links fast-Ethernet interface **13** with a fault-tolerant transceiver **15**. Fault tolerant transceiver **15** establishes a connection between connection **30** and one of two connections **29** and **31** to respective fast-Ethernet switches **19** and **17** (these "switches" as used herein are SNMP managed network Switches). Switches **17** and **19** are connected in a fault-tolerant matter to network **21** through connections **23** and **25**.

Fault-tolerant transceiver **15** may be purchased from a number of vendors including, for example, a Digi brand, model MIL-240TX redundant port selector; while fast-Ethernet switches **17** and **19** may also be purchased from a number of vendors and may include, for example, a Cisco brand, model 5000 series fast-Ethernet switch.

Operationally, traffic normally passes from fast-Ethernet interface **13** through fault-tolerant transceiver **15**, and over a primary connection **29** or **31** to respective switch **17** or **19** and on to network **21**. The other of connections **29** and **31** remains inactive. Network **21** and switches **17** and **19** maintain routing information that directs traffic bound for computer **11** through the above-described primary route.

In the event of a network connection failure, fault-tolerant transceiver **15** will switch traffic to the other of connection **29** and **31**. For example, if the primary connection was **31**, and connection **31** broke, fault-tolerant transceiver **15** would switch traffic to connection **29**.

When, for example, traffic from computer **11** begins passing over its new, backup connection **29** through switch

19, network routing has to be reconstructed such that traffic bound for computer **11** is routed by the network to the port on switch **19** that connection **29** is attached to. Previously, the routing directed this traffic to the port on switch **17** that connection **31** was attached to.

Several problems arise from the above-described operation. First, the rebuilding of network routing to accommodate passing traffic over the back-up connection may take an extended period of time. This time may range from seconds to minutes, depending upon factors including network equipment design and where the fault occurs. Second, fault-tolerant transceiver **15** is only sensitive to a loss of the physical receive signal on the wire pair from the switches (e.g., **17** and **19**) to the transceivers. It is not sensitive to a break in the separate wire pair from the transceiver to the switch. Also, it is sensitive only to the signal from the switch to which it is directly attached and does not test the backup link for latent failures which would prevent a successful recovery. This technique also fails to test the switches themselves.

Another example of a previous technique for connecting a computer **11** to a network **21** is shown in FIG. 2. Network switches **17** and **19** and their connection to each other and network **21** is similar to that shown in FIG. 1. However, in this configuration, each of switches (e.g., **17** and **19**) connects to its own fast-Ethernet interface (e.g., **13** and **14**) within computer **11**.

Operationally, only one of interfaces **13** and **14** is maintained active at any time. When physical signal is lost to the active interface, use of the interface with the failed connection is ceased, and connectivity begins through the other, backup interface. The backup interface assumes the addressing of the primary interface and begins communications. Unfortunately, this technique shares the same deficiencies with that depicted in FIG. 1. Rerouting can take an extended period of time, and the only failure mode that may be detected is that of a hard, physical connection failure from the switch to the transceiver.

The present invention is directed toward solutions to the above-identified problems.

SUMMARY OF THE INVENTION

Briefly summarized, in a first aspect, the present invention includes a method for managing network routing in a system including a first node, a second node and a third node. The first node has primary and secondary connections to the second and third nodes, respectively. Also, the second and third nodes are connected by a network.

The method includes periodically communicating between the first and the second or third node over at least the primary connection. A status of network connectivity between the communicating nodes is thereby determined.

If the network connectivity determined is unacceptable, roles of the primary and secondary connections are swapped to establish new primary and secondary connections. A message is then sent with an origin address of the first node to the second node over the new primary connection. The origin address of this message facilitates the network nodes learning about routing to the first node over the new primary connection.

As an enhancement, the first node may include a first port connected to the primary connection and a second port connected to the secondary connection. The first and second ports have first and second network addresses, respectively; and the first node has a system network address. The periodic communication may be transmitted from the first

port of the first node with an origin address of the first port. Further, the origin address of the message sent if network connectivity was unacceptable may be the system network address of the first node. Also, the periodic communication may be a ping message having the first network address of the first port as its origin address. This ping message may be destined for the second or third node.

If the ping message fails, another ping message may be sent from the second port to the other of the second and third nodes, not previously pinged. If this ping message is successful, the method may include swapping the roles of the primary and secondary connections and pinging the second node over the new primary link.

As yet another enhancement, the status of the connection between the second port and the other of the second and third nodes to which the previous ping was sent is determined.

In another aspect, the present invention includes a system for implementing methods corresponding to those described hereandabove. In this embodiment a link manager may be attached to the computer and may provide connectivity between the computer and the primary and secondary connections. As implementation options, the link manager may be, for example, integral with the computer (e.g., on a main board of the computer), on an expansion board of the computer, or external to the computer. Also, the computer may be an operator workstation or a controller such as, for example, an industrial or environmental controller.

BRIEF DESCRIPTION OF THE DRAWINGS

The subject matter regarded as the present invention is particularly pointed out and distinctly claimed in the concluding portion of the specification. The invention, however, both as to organization and method of practice, together with further objects and advantages thereof, may best be understood by reference to the following detailed description taken in conjunction with the accompanying drawings in which:

FIGS. **1–2** depict prior art systems for managing fault-tolerant network connections;

FIG. **3** depicts a fault-tolerant network connection topology in accordance with one embodiment of the present invention;

FIG. **4** is a functional block diagram of the link manager of FIG. **3** in accordance with one embodiment of the present invention;

FIGS. **5–7** are flow-diagrams of techniques in accordance with one embodiment of the present invention; and

FIGS. **8–11** depict several topologies in conformance with the techniques of the present invention.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

In accordance with the present invention, depicted herein are techniques for establishing a fault-tolerant connection to a network that overcome the disadvantages of prior techniques discussed hereinabove. That is, according to the present invention, connectivity problems are quickly detected, and upon assumption of an alternate (back-up) connection, network reroute times are mitigated.

Turning to FIG. **3**, a fast-Ethernet interface **13** is connected to both a link manager **33** and a CPU **31**. The topological relationship between fast-Ethernet interface **13**, link manager **33** and CPU **31** will vary with implementation requirements. Several example topologies are discussed hereinbelow in regard to FIGS. **9–11**; however, many other topologies will become apparent to those of ordinary skill in the art in view of the disclosure herein.

The techniques disclosed herein are not limited to fast-Ethernet technology. Other networking technologies may be subjected to the techniques disclosed herein, such as, for example, conventional Ethernet technology.

Link manager **33** is connected to both fast-Ethernet interface **13** and CPU **31**. The connection to fast-Ethernet interface **13** is that which would be normally used for network connectivity. The connection of link manager **33** to CPU **31** is for configuration and control purposes. In one implementation example, link manager **33** and fast-Ethernet interface **13** may each be PCI cards within a personal computer architecture. In this example, their connections to CPU **31** are by way of the PCI bus. A cable may connect fast-Ethernet interface **13** and link manager **33**.

Two network connections **29** and **31** (for example, fast-Ethernet connections) couple link manager **33** to switches **19** and **17**, respectively. Connections **23** and **25** couple switches **17** and **19** to network **21**, which connects them to each other.

Link manager **33** is more specifically depicted in FIG. **4**. A fast-Ethernet interface **41** provides connectivity (e.g., PCI bus interface) with an attached host computer. Computer interface **45** also attaches to the host computer and facilitates configuration and control of link manager **33**. Fast-Ethernet interfaces **47** and **49** provide redundant network connectivity. Lastly, logic **43** interconnects the above-described elements. In a preferred embodiment, logic **43** is implemented as an ASIC; however, the particular implementation of logic **43** will vary with product requirements. In other implementation examples, logic **43** could be implemented using a programmed processor, a field programmable gate array, or any other form of logic that may be configured to perform the tasks disclosed therefor herein.

To briefly summarize, the techniques of the present invention send test messages across each connection of the link manager to the attached switches. A break in a connection, or faulty connection, is detected upon a failed response to one of the test messages. In response to this failure, traffic is routed across the remaining good connection. To facilitate fast protocol rerouting, a test message is sent across the now active connection bound for the switch connected to the inactive connection. This message traverses both switches causing each to learn the new routing. Rerouting is therefore accomplished quickly.

More particularly, according to one-embodiment, FIGS. **5–6** depict flow-diagrams of operational techniques in accordance with one embodiment the present invention. To begin, the link manager pings a switch connected to the primary, active connection, every T_p seconds, STEP **101**. The ping message contains a source address unique to the link manager port currently associated with the active connection. If the active connection is ok, pinging thereof continues, STEP **101**. Also, a check is regularly performed to detect a loss of receive signal on the active connection interface, STEP **113**.

If either pinging fails on the active connection, or carrier has been lost, a test is performed to check whether the back-up connection status is good, STEP **105**. If the back-up connection is unavailable, no further action can be taken and pinging of the primary connection continues in anticipation of either restoration of the active connection or availability of the back-up connection. Also under this condition, the host computer may be notified such that it may take appropriate action, such as, e.g., to enter a fail-safe condition.

If the back-up connection status is good, the link manager is configured to direct traffic through the back-up

connection, STEP 107. Further, a ping message is sent from the link manager, through the switch connected to the back-up connection and to the switch connected to the primary, failed, connection, STEP 109. This ping message contains a source address of the computer connected to the link manager. As a result, the switches connected to the primary and back-up connections are made aware of the new routing to the computer. This facilitates the immediate routing of traffic bound for the computer over the back-up, secondary, connection. Lastly, the roles of active and backup connections are swapped and the process iterates, STEP 111.

Turning to FIG. 6, a flow-diagram depicts a technique for maintaining the status of the back-up connection. A ping is send over the back-up connection to its respective switch every T_p seconds, STEP 115. The ping message contains a source address unique to the link manager port currently associated with the backup connection. If the back-up connection is good, that is, the ping is responded to timely, STEP 117, then the back-up connection status is set to good, STEP 119. If the response to the ping message is not timely received, the back-up connection status is set to bad, STEP 121 (A maintenance alert may also be generated. The invention facilitates detecting latent faults in unused paths and repairing them within the MTBF of a primary fault.) In either case, the processor iterates to the pinging step, STEP 115.

According to the above-described embodiments ping messages are sent from the link manager, across each connection to the switch attached thereto. Failure of these ping messages will indicate failure of the link the ping message was sent across. In accordance with the embodiment of FIG. 7 described below, ping messages are sent across each link, but are bound for the switch connected to the other connection. Thus, the ping message must traverse one switch to get to the destination switch, traversing both the connection from the link manager to the immediately attached switch and across the connection between the switches. Thus, the technique described below can localize faults in the connections between the link manager and each switch and the connection between the switches. Further, this embodiment contains example information on how timed message transmission can be implemented using a common clock.

As described above, the pings sent from each port have a unique source address for that particular port. However, to facilitate fast rerouting, the final ping, once the port roles are swapped uses the source address of the attached computer system.

To begin, a clock tick is awaited, STEP 201. Clock ticks are used as the basis for timing operations described herein. If a clock tick has not occurred, no action is taken. However, if a clock tick has occurred a first counter is decremented, STEP 203. This first counter is designed to expire, on a 0.5 second basis (of course, this time can be adjusted for particular application requirements).

If the first counter expired, indicating that the 0.5 second period has elapsed, a ping message is sent from the active port to the standby switch using the address of the active port, STEPS 205, 207. If the ping is successful, STEP 209, a second counter with a 30 second interval is decremented, STEP 211. The second counter decrement is also performed if the first counter decrement did not result in the 0.5 second time period expiring, STEP 205. If the second counter has not expired, STEP 213, then the process iterates awaiting a next clock tick, STEP 201. If the second counter has expired, a ping is sent from the standby port to the active switch using the standby port's address, STEP 215. If the ping was successful, STEP 217 then the process iterates awaiting another clock tick, STEP 201.

If the ping from the active port to the standby switch failed, STEP 209, a ping is sent from the standby port to the active switch, STEP 227. If this ping is successful, STEP 229, then the roles of the active and standby ports and switches are reversed, STEP 231, and a ping is sent from the now active port to the now standby switch using the address of the computer station, STEP 233. This ping facilitates the switches learning the new path to the computer thus correcting routing information. Furthermore, the old active port is determined to be in error, STEP 235.

Turning back to STEP 215, if the ping from the standby port to the active switch failed (STEP 217) a ping is sent from the active port to the standby switch, STEP 219. If this ping fails, there is an error associated with the standby port, STEP 223.

Turning back to STEP 227, a ping was sent from the standby port to the active switch. If this ping failed, then the current error must be associated with either the switches, the network between the switches or both ports may be bad. Therefore, for the following steps, it is most helpful to refer to the ports and switches as the "A port", "A switch", "B port" and "B switch", wherein the A port is directly connected to the A switch and B port is directly connected to the B switch. The notion of which port is currently active and which port is currently backup is not significant to the following steps.

Again, if the ping from the standby port to the active switch, STEPS 227, 229, failed then a ping is sent from the A port to the A switch, STEP 237. If this ping is successful, STEP 239, then the A port is set as the active port, STEP 241. A ping is then sent from the B port to the B switch, STEP 243. If this ping failed, STEP 245, then the error is associated with B switch, STEP 247; however, if the ping was successful, then the error is associated with the network, STEP 249.

If the ping from the A port to the A switch, STEP 237, failed, STEP 239, then the B port is set as active, STEP 251. A ping is then sent from the B port to the B switch, STEP 253. If this ping failed, then an error is associated with both ports, STEP 259; however, if the ping was successful, STEP 255, then the error is associated with the A switch, STEP 257.

In each of the above steps, once the error is determined and set (STEPS 223, 235, 247, 249, 257, and 259), an interrupt is sent to the host processor (STEP 255) for providing notification of the change in network configuration.

The techniques of the present invention may be implemented in different topologies. As examples, several of these topologies are depicted in FIGS. 8–11.

In each of the examples, the computer depicted may be, for example, a workstation, an embedded processor, a controller, (e.g., industrial or environmental) or other computer type.

Beginning with FIG. 8, a computer 11 is depicted and contains fast-Ethernet interface 13 and link manager 33 connected by cable 30. Connections 29 and 31 couple the system to a network. The particular implementation and use of computer 11 will vary. In one example, computer 11 is a PCI bus-based computer and fast Ethernet interface 13 and link manager 33 are PCI interface cards. In another embodiment, all circuitry may be on a common board (e.g., the system motherboard).

In FIG. 9, the functions of link manager 33 and fast-Ethernet interface 13 have been integrated onto a single interface card. As one example, this card may interface with its host computer using a PCI bus.

In FIG. 10, fast-Ethernet interface 13 is incorporated on a main board (e.g., a motherboard) of computer 11. Link manager 33 is a peripheral (e.g., PCI) interface card.

In FIG. 11, fast-Ethernet interface 13 may be incorporated on a main board of computer 11 or as a separate interface card. Link manager 33 is disposed external to computer 11 and is connected thereto by connections 30 and 63. Connection 63 is particularly used for command and control of link manager 33 and interfaces with computer 11 through a communications port 61 (e.g., a serial or parallel port).

A variety of techniques are available for implementing the techniques described herein. The present invention is not meant to be limitive of such implementation, as many options are available to those of ordinary skill in the art and will be apparent in view of the disclosure herein. Implementations may take form of software, hardware, and combinations of both. Dedicated logic, programmable logic, and programmable processors may be used in the implementation of techniques disclosed herein. One particular implementation example using programmable logic to implement a simple instruction set capable of implementing the techniques described herein is described in detail in Appendix A, "HDS 5608-Dual Switched Ethernet Interface, Revision 1.1" attached hereto and incorporated by reference herein in its entirety.

While the invention has been described in detail herein, in accordance with certain preferred embodiments thereof, many modifications and changes thereto can be affected by those skilled in the art. Accordingly, is intended by the appended claims to cover all such modifications and changes as fall within the true spirit and scope of the invention.

We claim:

1. A method for managing network routing in a system including a first node, a second node, and a third node, wherein said first node has a primary connection to said second node and a secondary connection to said third node, wherein said second node and said third node are connected by a network, and wherein said method includes:

(a) periodically communicating between said first node and one of said second node and said third node over at least said primary connection and thereby determining a status of network connectivity between said first node and said one of said second node and third node; and

(b) if said network connectivity status determined in said step (a) is unacceptable, swapping roles of said primary and said secondary connections to establish new primary and secondary connections and sending a message with an origin address of said first node to said second node over said new primary network connection, wherein said origin address of said message facilitates said network nodes learning about routing to said first node over said new primary connection.

2. The method of claim 1, wherein said first node includes a first port connected to said primary connection and a second port connected to said secondary connection, said first port having a first network address, said second port having a second network address and said first node having a system network address, wherein said periodic communication is transmitted from said first port of said first node with an origin address of said first port.

3. The method of claim 2, wherein said origin address of said sending said message of said step (b) comprises said system network address of said first node.

4. The method of claim 3, wherein said periodic communication between said first node and one of said second node and said third node comprises a ping message having said first network address of said first port as an origin address of said ping message.

5. The method of claim 4, wherein said ping message has a destination of said second node.

6. The method of claim 4, wherein said ping message has a destination of said third node.

7. The method of claim 4, wherein if said ping fails, a ping is sent from said second port to the other of said second node and said third node.

8. The method of claim 7, wherein if said ping from said second port to said other of said second node and said third node is successful, said method includes performing said swapping roles of said primary and secondary connections and said pinging of said second node over said new primary link of said step (c).

9. The method of claim 2, further comprising sending a ping message from said second port, with an origin address thereof, to the other of said second node and said third node to determine a status of network connectivity thereto.

10. A method for managing network routing in a system including a computer, a first network switch, and a second network switch, said first and second network switches being network connected, wherein said computer has an active connection to said first network switch and a backup connection to said second network switch, said method including:

(a) periodically pinging said second network switch by transmitting a ping message bound for said second network switch over said active connection, said ping having an address of a port of said computer connected to said active connection; and

(b) if said ping fails, and said backup connection is available, swapping roles of said active and backup connections to establish new active and backup connections and sending a ping with an origin address of said computer system to said first network switch over said new active connection, wherein said origin address of said ping facilitates said network nodes learning about routing to said computer over said new active connection, said address of said computer system being different than said address of said port.

11. A system for managing network routing including a first node, a second node, and a third node, wherein said first node has a primary connection to said second node and a secondary connection to said third node, said system including:

(a) means for periodically communicating between said first node and one of said second node and said third node over at least said primary connection and determining a status of network connectivity between said first node and said one of said second node and third node thereby;

(b) means for determining if said network connectivity status determined in said step (a) is unacceptable, and if so, for swapping roles of said primary and said secondary connections to establish new primary and secondary connections and for sending a message with an origin address of said first node to said second node over said new primary network connection, wherein said origin address of said message facilitates said network nodes learning about routing to said first node over said new primary connection.

12. The system of claim 11, wherein said first node comprises a computer.

13. The system of claim 12, further including a link manager attached to said computer, said link manager providing connectivity between said computer and said primary and secondary connections.

14. The system of claim 13, wherein said link manager is integral with said computer.

15. The system of claim 14, wherein said link manager is on a main board of said computer.

16. The system of claim 13, wherein said link manager is on an expansion board of said computer.

17. The system of claim 13, wherein said link manager is external to said computer.

18. The system of claim 12, wherein said computer comprises an operator workstation.

19. The system of claim 12, wherein said computer comprises one of an industrial controller and an environmental controller.

* * * * *

US006163529A

United States Patent [19]

Nagel et al.

[11] Patent Number: 6,163,529

[45] Date of Patent: Dec. 19, 2000

[54] **TRANSMISSION SYSTEM WITH LINE STATUS DETERMINATION INDEPENDENT OF RECEIVED PACKET ORDER**

[75] Inventors: **Peter Nagel**, Erlangen; **Hans-Georg Keller**, Nürnberg, both of Germany

[73] Assignee: **U.S. Philips Corporation**, New York, N.Y.

[21] Appl. No.: **08/576,539**

[22] Filed: **Dec. 21, 1995**

[30] **Foreign Application Priority Data**

Dec. 23, 1994 [DE] Germany 44 46 248

[51] **Int. Cl.**[7] **H04Q 1/20**

[52] **U.S. Cl.** **370/244**

[58] **Field of Search** 370/13, 14, 15, 370/16, 60.1, 94.2, 110.1, 111, 216, 217, 218, 241, 242, 244, 250, 395, 410, 522, 528; 340/825.16, 825.17; 371/20.1, 20.2, 42, 46, 47.1

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,321,688	6/1994	Nakano et al.	370/110.1
5,343,462	8/1994	Sekihata et al.	370/60.1
5,461,607	10/1995	Miyagi et al.	370/16
5,475,696	12/1995	Taniguchi	371/42
5,553,057	9/1996	Nakayama	370/60.1

FOREIGN PATENT DOCUMENTS

0472408A2	2/1992	European Pat. Off. .
0518199A2	12/1992	European Pat. Off. .

OTHER PUBLICATIONS

"ATM—Die Technik des Breitband-ISDN", by Gert Sigmund, R v. Decker's Verlag, G. Schenck, Heidelberg, 1993, pp. 133 to 137.

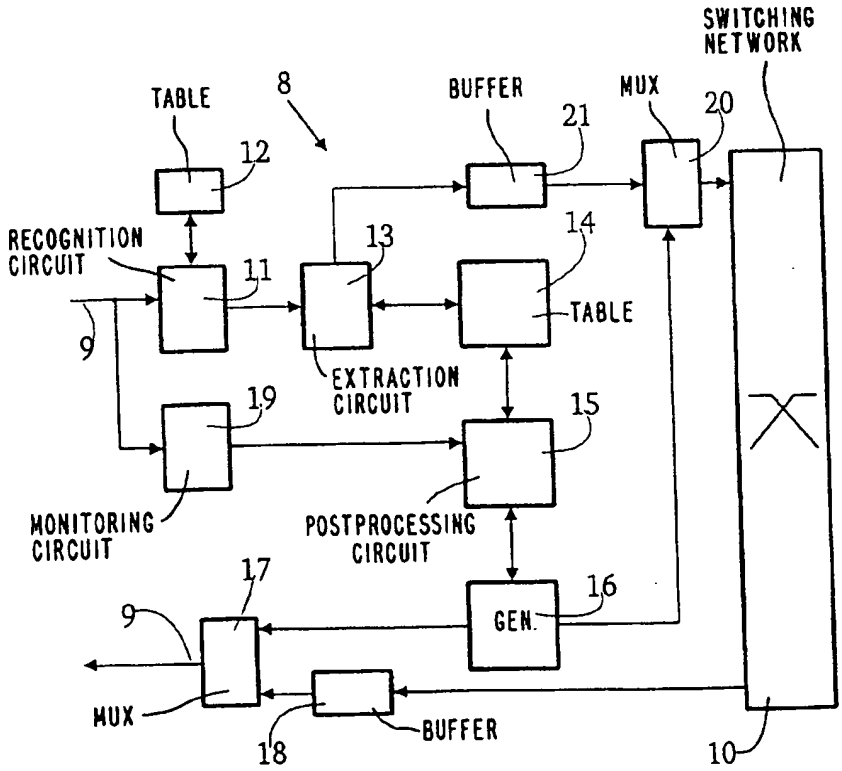
Primary Examiner—Chau Nguyen

Attorney, Agent, or Firm—David R. Treacy

[57] **ABSTRACT**

A transmission system comprises a plurality of network nodes which for receiving and transmitting packets which contain connection-related status information about the transmission system. A network node comprises at least one evaluation unit which stores status information extracted from a packet in a table. Furthermore, irrespective of the order of the received packets, the evaluation unit determines current, connection-related statuses in a given order for each connection, based upon the statuses also stored in the table and determined thus far, as well as on the received connection-related status information signals.

4 Claims, 1 Drawing Sheet



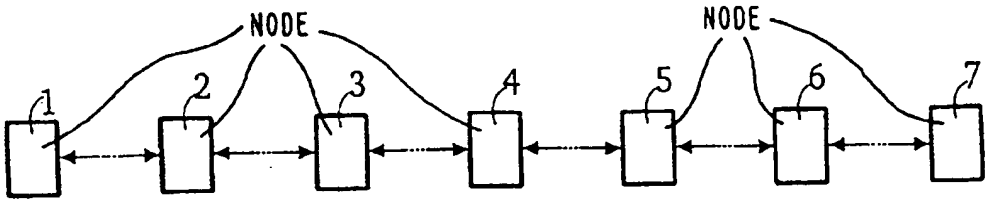

NODE
NODE
1
2
3
4
5
6
7

FIG. 1

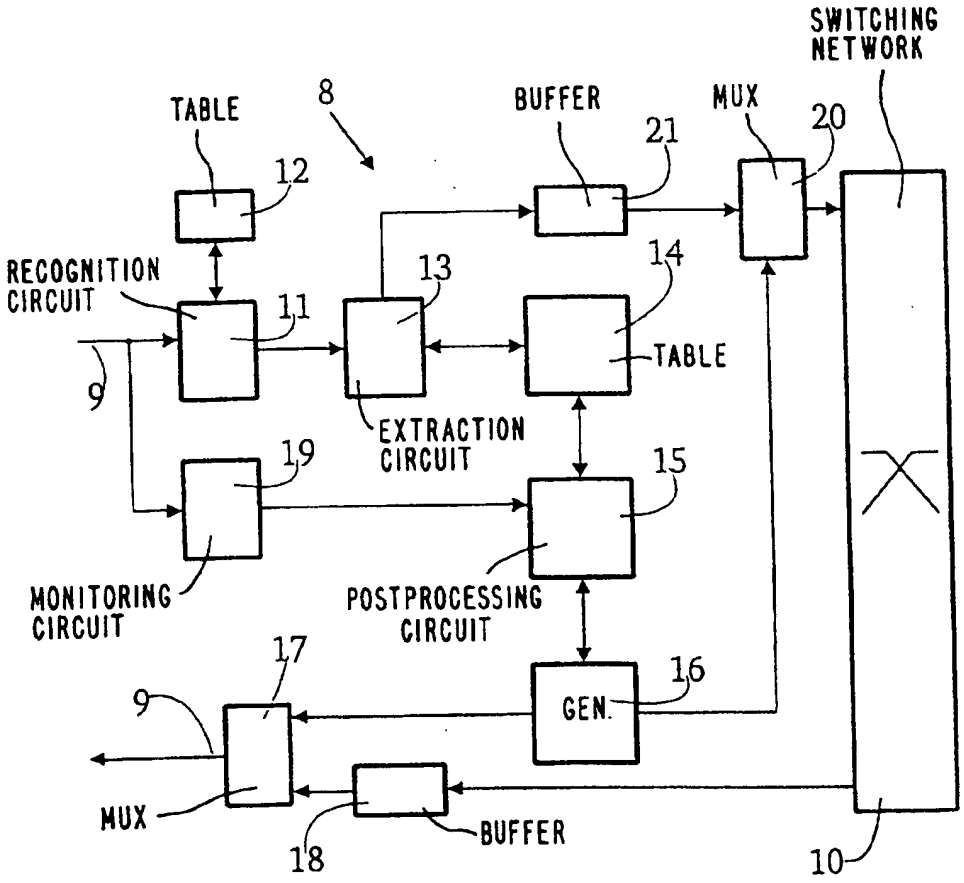

SWITCHING NETWORK
TABLE
8
BUFFER
MUX
12
21
20
RECOGNITION CIRCUIT
13
14
11
TABLE
9
EXTRACTION CIRCUIT
19
15
MONITORING CIRCUIT
POSTPROCESSING CIRCUIT
17
16
9
GEN.
MUX
BUFFER
18
10

FIG. 2

TRANSMISSION SYSTEM WITH LINE STATUS DETERMINATION INDEPENDENT OF RECEIVED PACKET ORDER

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to a transmission system comprising a plurality of network nodes which are provided each for receiving and transmitting packets which contain connection-related status information signals about the transmission system.

2. Discussion of the Related Art

Such a transmission system comprising a plurality of network nodes is known from the title "ATM—Die Technik des Breitband-ISDN" by Gert Sigmund, R v. Decker's Verlag, G. Schenck, Heidelberg, 1993, pp. 133 to 137. Such network nodes may be, for example, switching centres, cross-connects or regenerative repeaters. Data or information signals are then transmitted between the network nodes in fixed-length packets (cells) in the asynchronous transfer mode. The packets or cells may then be transported in a synchronous transport module (STM) which forms part of a signal according to the standard of the synchronous digital hierarchy (SDH) or according to the standard of the synchronous optical network (SONET). A cell consists of a 5-byte-long header field which contains control information, and a 48-byte-long information field. The control information in the header field of a cell contains a Virtual Path Identifier (VPI) which indicates a virtual path (VP), and a Virtual Connection Identifier (VCI) within a virtual path, which indicates a Virtual Connection (VC). A plurality of VCs are combined to become a virtual path.

To ensure proper operation, to ensure that error conditions are signalled and monitor information signals (for example, with respect to the availability and efficiency of the system) are transported, cells carrying OAM data (OAM cells) in their information field are exchanged between the network nodes (OAM=Operations Administration and Maintenance). How such OAM cells are evaluated in a network node is not known from above title.

SUMMARY OF THE INVENTION

Therefore, it is an object of the invention to provide a transmission system comprising a plurality of network nodes in which OAM cells are evaluated.

A transmission system of the type defined in the opening paragraph achieves the object in that a network node comprises at least one evaluation unit which:

stores status information signals extracted from a packet in a table and

irrespective of the order of the received packets, determines current, connection-related statuses in a given order for each connection based upon the statuses also stored in the table and determined thus far, as well as on the received connection-related status information signals.

According to the invention a network node includes an evaluation unit for one line, which unit determines what type of packet or cell is present. It is possible that a network node includes further evaluation units for further lines. If a packet contains connection-related status information signals, it will be an OAM cell. Such status information signals may be, for example, AIS or RDI information signals in the information field of a cell. The AIS (AIS=Alarm Indication Signal) information signal indicates, for example, that there is a line defect, and the RDI (RDI=Remote Defect Indication) information signal, for example, that there is a line defect in the opposite direction. These status information signals and further data (VPI, VCI and so on) are stored in a table.

In addition to this current cell processing, the evaluation unit carries out a postprocessing or background processing. During this postprocessing, stored data are taken from the table for a VP or VC and at least one current status is computed. Such a status indicates, for example, that a line defect has occurred. A presupposition is then that such a status information signal (for example, AIS information) has been announced by OAM cells. If one or more current statuses have been determined for a specific VP or VC, first the current statuses of all the other VPs or VCs are evaluated before the current statuses of above-mentioned specific VP or VC are determined again.

When the statuses are computed, it may happen that OAM cells must be generated by the evaluation unit and sent. As a result of said type of postprocessing, cells of a specific VP or VC are sent a certain period of time apart, so that aregular cell stream develops and the danger of a cell jam is reduced.

To evaluate cells that have been transmitted in the asynchronous transfer mode, the evaluation unit includes a recognition circuit at least for recognizing the type of cell, the Virtual Path (VP) and—if available—the Virtual Connection (VC), and an extraction circuit. At least for specific OAM cells, this extraction circuit is arranged for extracting status information signals and for transferring the status information signals to the table.

The evaluation unit further includes a postprocessing circuit which periodically extracts data relating to a connection from the table, determines and stores in the table current, connection-related statuses, and, in the case of certain statuses, sends a control command to a further generator circuit to have it generate OAM cells, which further generator circuit is included in the evaluation unit. One period corresponds to the duration necessary for computing the statuses of all the connections. This is understood to mean that all possible VPs and VCs are processed consecutively in equidistant time slots.

The evaluation unit includes a generator circuit which generates OAM cells. These cells are fed either to a switching network in the network node or to a line. The cells which are to be fed to the line must first be applied to a multiplexer which receives cells either coming from the switching network or coming from the generator circuit. The cells coming from the generator circuit have priority. Therefore, cells simultaneously arriving from the switching network must be buffered in a buffer store included in the evaluation unit. The evaluation unit further includes another multiplexer which transfers either cells coming from the extraction circuit, or priority cells coming from the generator circuit to the switching network. For buffering the cells coming from the extraction circuit, the evaluation unit includes a further buffer store. Because the cells in the generator circuit are generated regularly and in a controlled manner, the buffer stores may be small-capacity buffer stores.

In a further embodiment of the invention a monitoring circuit included in the evaluation unit is provided at least for monitoring whether the connected line has a line defect and for passing on the information about the line defect to the postprocessing circuit. Such a line defect is also taken into account by the postprocessing circuit. This postprocessing circuit causes the generator circuit to generate OAM cells which contain AIS or RDI information signals.

The table further contains an entry for each VC, which entry refers to the associated VP. As a result, it is possible to simply form statuses of the VCs in response to statuses of the associated VP.

These and other aspects of the invention will be apparent from and elucidated with reference to the embodiments described hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

In the drawings:

FIG. 1 shows part of a transmission system comprising network nodes, and

FIG. 2 shows a more detailed representation of a network node.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

FIG. 1 shows part of a transmission system comprising seven network nodes 1 to 7. Such network nodes may be, for example, switching centres, cross-connects or regenerative repeaters. Information signals are transmitted in the transmission system in fixed-length packets (cells) in the asynchronous transfer mode. A cell consists of a 5-byte-long header field which contains control information and a 48-byte-long information field. The cells can meanwhile be shifted in a synchronous transport module (STM). A synchronous transport module is part of a signal according to the synchronous digital hierarchy (SDH) standard, or the synchronous optical network (SONET) standard.

There may be a virtual connection or a virtual path between the network nodes 1 and 7. A Virtual Connection is referenced a VC. A plurality of VCs are combined to a Virtual Path which is referenced a VP. A VP is featured by a Virtual Path Identifier VPI and a VC is featured by a Virtual Connection Identifier VCI plus the respective virtual path identifier VPI in the header field of a cell. These identifiers may be changed in various network nodes, for example, when a cell goes from the network of one system operator to the network of another system operator.

To ensure proper operation, the signalling of error conditions and the transport of monitor information signals (for example, relating to the availability and the efficiency of the system), cells which contain OAM data (OAM cells) in their information field, are exchanged between the network nodes (OAM=Operations Administration and Maintenance). Therefore, the network nodes have an evaluation unit for evaluating such OAM cells.

An embodiment for such an evaluation unit 8 is shown in FIG. 2 which unit is connected between a line 9 and a switching network 10. Instead of a switching network, it is also possible that a terminal unit is connected when the network node is located at the end of the connection (for example, network nodes 1 and 7 in FIG. 1). The evaluation unit 8 comprises a recognition circuit 11 and a first table 12 by which two elements the VPs and VCs and the type of cell are determined. The recognition circuit 11 learns the Virtual Connection Identifier VCI and the Virtual Path Identifier VPI from the header field of a cell supplied via line 9 and applies these identifiers to table 12. Table 12 supplies the VP and—if available—an associated VC to the recognition circuit 11. Furthermore, the recognition circuit 11 includes a decoder which determines the type of cell on the basis of data of the header field. This decoder in the recognition circuit 11 evaluates the Virtual Path Identifier VPI, the Virtual Connection Identifier VCI, a Payload Type Identifier PTI and a bit in the header field, which bit is referenced CLP (Cell Loss Priority) and indicates the priority of the cell.

The decoder determines the type of cell according to the following table:

Type of cell	VPI	VCI	PTI	CLP
Payload	XXX	>001F	0xx	x
F4SEG	XXX	0003	0x0	x
F4ETE	XXX	0004	0x0	x
F5SEG	XXX	zzzz	100	x
F5ETE	XXX	zzzz	101	x

The decoder thus detects whether a payload cell or an OAM cell of the type F4SEG, F4ETE, F5SEG or F5ETE is present. In the case of a cell of the type F4SEG or F5SEG, the cell contains OAM data of only one section of a connection. OAM data about the end-to-end connection are available in either of the other two types of cells F4ETE, F5ETE. Further, the cell types F4SEG and F4ETE relate to a VP and cell types F5SEG and F5ETE to a VC. Other types of cells which may occur may be featured either as payload data or as an irrelevant type of cell. The figure X occurring in the table represents a hexadecimal number, the figure x a binary number and the figure Z a hexadecimal number except for 0. The table mentioned above is taken from the standardization proposal ITU-T-I.610.

The recognition circuit 11 transfers the buffered cell, the data produced by the table 12 and the type of cell recognized by the decoder to an extraction circuit 13 also included in the evaluation unit 8. The extraction circuit 13 extracts data (status information) from the information field of the received cell only if the type of cell is F4SEG, F4ETE or F5SEG or F5ETE. In the other cases the cell buffered in the extraction circuit 13 is transferred without data being extracted therefrom.

From the information field of a cell is extracted, for example, AIS information (AIS=Alarm Indication Signal), or RDI information (RDI=Remote Defect Indication) in the extraction circuit 13. The AIS information indicates, for example, that there is a line defect and the RDI information indicates, for example, that there is a line defect in the opposite direction. The extraction circuit 13 is coupled to a table 14 which has specific entries for all the VPs and VCs. The extraction circuit 13 extracts specific data from the table 14 which are stored in the VP and VC of the arrived cell. Each VP and VC is assigned status information announcing whether AIS information, RDI information or other information has arrived. Depending on optionally available AIS, RDI, VPI, VCI or PTI information, the type of cell and further information signals, the data of the VPs and VCs are processed and returned to the table 14. If there is no VC (network nodes only for VPs), the processing steps necessary for the VCs are omitted.

A postprocessing circuit 15 which forms part of the evaluation unit 8, as does table 14, is coupled to the table 14. The postprocessing circuit 15 periodically extracts from the table 14 data which relate to a VP or VC and processes these data. This means that in the postprocessing circuit 15 the data of all the VPs and VCs are processed consecutively.

There is assumed, for example, that there are two VPs where VPI=1 and VPI=2, and three VCs where VPI=1/VCI=

10, VPI=1/VCI=11 and VPI=2/VCI=12. The table **14** then shows, for example, the following entries:

Address	Type	VPI	VCI	VP_REF	...
0	VP	1	—	—	
1	VP	2	—	—	
2	VC	1	10	0	
3	VC	1	11	0	
4	VC	2	12	1	
:					
:					
:					

Each connection in this table is assigned to an address. The table contains at least the type of connection (VP or VC), VPI, VCI and a reference entry VP_REF which entry features the associated VP for a VC. Instead of one common table it is also possible that two separate tables are used for all the VPs and also for all the VCs.

In the postprocessing circuit **15** first the data of the VP, with VP=1, buffered in table **14** are processed and, after being processed, these data are returned to table **14**. The VP, with VPI=1, will not be re-processed by the postprocessing circuit **15** until this circuit has processed the VP, where VPI=2, and the VCs where VPI=1/VCI=10, VPI=1/VCI=11 and VPI=2/VCI=12, as well as all further entries of table **14**.

In the postprocessing circuit **15** there is verified, for example, whether a line defect, AIS information and so on is present and whether this is to be announced via line **9** to a network node by means of an OAM cell which contains RDI information or via the switching network **10** by an OAM cell which contains AIS information.

For example, a line defect on line **9** is detected by a monitoring circuit **19** which then announces this fact to the postprocessing circuit **15**. It is also possible that an error of line **9** is detected and announced by an external facility. OAM cells which contain AIS or RDI information are generated by a generator circuit **16** which thereto receives the indication from the postprocessing circuit **15**. The generator circuit **16** then forms an OAM cell. The header field of this cell is filled with the appropriate information signals from the table **14** and the AIS or RDI information is inserted into the information field of the OAM cell.

If the newly formed OAM cell contains RDI information, the latter is transferred from the generator circuit **16** to a multiplexer **17** which immediately inserts the OAM cell into the cell stream of line **9**. An appropriate control command for the multiplexer **17** is generated by the generator circuit **16**. If, meanwhile, cells were sent from switching network **10** to line **9**, they are buffered in a buffer store **18** and transferred to line **9** at a later stage via multiplexer **17**. An OAM cell which contains AIS information, is applied to the switching network **10** via a further multiplexer **20** controlled by the generator circuit **16**. Cells coming from the extraction circuit **13** are buffered in a buffer store **21** which is connected between extraction circuit **13** and multiplexer **20**. The buffer store **21** may also be coupled to the input of the recognition circuit **11** instead of to the multiplexer **20**.

Various connection-related independent processes run in the extraction circuit **13** and the postprocessing circuit **15**. The table **14** is accessed then. As discussed above, status information signals and further statuses derived from the status information signals (for example, AIS status of the connection) are stored in the table.

The AIS information signals are processed via a process. The table **14** contains one memory cell for each connection which is referenced AISD and denotes whether this connection is in the AIS status (AISD>0). AISD represents a binary coded integer. The AIS status is adopted if a single OAM cell which contains AIS information has arrived. This is detected by the extraction circuit **13** and in table **14** the memory cell AISD for this connection is set, for example, to the value **6**. The AIS status is abandoned when a single cell of this connection arrives that contains payload. This is also detected by the extraction circuit **13** which resets in this case the memory cell AISD to the value 0. The AIS status is also abandoned if for a given period of time no further OAM cells containing AIS information for this connection has arrived. This period of time is 2.5s±0.5s according to the current standardization proposal (ITU-T I.610). All further subsequent time conditions relate to this standardization proposal ITU-T I.610. To achieve this, the postprocessing circuit **15** decrements the value of AISD for the connection currently to be processed by a fixed period (in this illustrative embodiment the period should be 0.5s). All the connections are processed in the postprocessing circuit in one period. The AIS status is announced, for example, to a network management not further shown here. This process discussed above may be represented by the following routine:

Routine in the extraction circuit **13** for VP.
If there is a cell type F4ETE which contains AIS information, then:
  AISD=6;
otherwise, if there is a payload cell:
  AISD=0;
Routine in the extraction circuit **13** for VC.
If there is a cell type F5ETE which contains AIS information, then:
  AISD=6;
otherwise, if there is a payload cell:
  AISD=0
Routine in the postprocessing circuit **15**:
If AISD>0, then:
  AISD=AISD-1;
otherwise:
  AISD=0;

A second process is used for evaluating the RDI information in table **14**. Table **14** contains one memory cell for each connection, which cell is referenced RDID and denotes whether this connection is in the RDI status (RDID>0). RDID represents a binary-coded integer. The RDI status is taken if a single OAM cell containing RDI information has arrived. This is detected by the extraction circuit **13** and the extraction circuit **13** sets the memory cell RDID for this connection in table **14**, for example, to the value **6**. The RDI status is abandoned if no further OAM cell containing RDI information for this connection has arrived for a predetermined period of time. This period of time is, as observed before, 2.5s±0.5s. To achieve this, the postprocessing circuit **15** decrements the value of RDID by a fixed period (in this illustrative embodiment the period should be 0.5s). The RDI status is announced, for example, to the network management. This process discussed hereinbefore, may be represented by the following routine:

Routine in the extraction circuit **13** for VP.
If there is a cell type F4ETE which contains RDI information, then:
  RDID=6;
Routine in the extraction circuit **13** for VC.
If there is a cell type F5ETE which contains RDI information, then:

RDID=6;
Routine in the postprocessing circuit **15**:
If RDI>0, then:
RDID=RDID-1;
otherwise:
RDID=0.

A third process is used for generating OAM cells containing an AIS information signal. Such an OAM cell is only generated at a network node that does not lie at the beginning or end of a VP or VC (network nodes **2** to **6** in FIG. **1**). In that case there is a respective entry in table **14**. The OAM cells are then generated if, for example, a line defect of line **9** has been detected, or if in a VC an AIS status of the associated VP has been detected. In this context there should be presupposed that for sending an OAM cell which contains an AIS information signal, a maximum of 0.5s must be available and consecutive OAM cells must be sent Is apart. Therefore, the duration of 0.5s is also selected as a period for the postprocessing.

In addition, a further status for each connection, which status is referenced AISN and indicates whether for this connection an OAM cell which contains an AIS information signal is to be generated by the generator circuit **16**, is stored in table **14**. AISN represents a binary-coded integer. The process may be described by the following illustrative routine for the postprocessing circuit **15**:
Routine in the postprocessing circuit **15**:
If either a line defect occurs or, in the case of a VC, there is an AIS status of the associated VP (AISD(VP_REF)>0), then:
If AISN=0, then:
AISN=(AISN+1)(mod2);
If there is a VP, then:
An OAM cell of the type F4ETE which contains AIS information is sent;
Otherwise:
An OAM cell of the type F5ETE which contains AIS information is sent;
otherwise
AISN=(AISN+1)(MOD2);
otherwise:
AISN=0.

In the case of a line defect, or a VC having the value AISD(VP_REF)>0, there is verified whether AISN is equal to 0. For determining AISD(VPREF), the respective table entry to which VP_REF refers is accessed and the associated value of AISD of the corresponding VP is read out.

If AISN is equal to 0, AISN is incremented modulo-2 and an OAM cell is formed. If there is a VP, the postprocessing circuit **15** causes the generator circuit **16** to generate an OAM cell of the type F4ETE which contains an AIS information signal. Otherwise—in the case of a VC—the generator circuit **16** is instructed to generate an OAM cell of the type F5ETE which contains an AIS information signal.

If AISN is not equal to 0, AISN is incremented modulo-2. No OAM cell is generated. The purpose of this is to adhere to the distance in time of Is between two consecutive OAM cells which contain an AIS information signal. If not 0.5s is chosen for this period, the modulo operator is to be modified accordingly.

If the conditions for generating an OAM cell are not satisfied, AISN is set to 0.

A further process is used for generating OAM cells which contain an RDI information signal. Such an OAM cell is generated only in a network node that lies at the beginning or end of a VP or VC (network nodes **1** and **7** in FIG. **1**). In that case, there is also an respective entry in table **14**. An OAM cell is generated if, for example, a line defect on line **9** has been detected, or an AIS status (AISD>0) has been recognized for this line, or an AIS status of the associated VP has been detected in a VC. There should be presupposed in this context that a maximum of 0.5s is available for sending an OAM cell which contains an RDI information signal and that consecutive OAM cells are to be sent 1s apart. Therefore, also the duration of 0.5s is chosen as a period for the postprocessing.

Furthermore, in table **14** there is one memory cell per connection vailable for the status referenced RDIN which indicates whether an OAM cell which contains an RDI information signal is to be generated for this connection by the generator circuit **16**. RDIN represents a binary-coded integer. This process can be described by the following routine by way of example for the postprocssing circuit **15**:
Routine in the postprocessing circuit **15**:
If either a line defect or an AIS status (AISD>0) occurs, or, in the case of a VC, there is an AIS status of the associated VP (AISD(VP_REF)>0), then:
If RDIN=0, then:
RDIN=(RDIN+1)(mod2);
If there is a VP, then:
An OAM cell of the type F4ETE which contains RDI information is sent; otherwise:
An OAM cell of the type F5ETE which contains RDI information is sent;
otherwise:
RDIN=(RDIN+1)(MOD2);
otherwise:
RDIN=0.

In the case of a line defect or an AIS status (AISD>0) or in the case of a VC having the value AISD(VPRF)>0, there is verified whether RDIN is equal to 0. If RDIN is equal to 0, RDIN is incremented modulo-2 and an OAM cell is formed. In the case of a VP, the postprocessing circuit **15** instructs the generator circuit **16** to generate an OAM cell of the type F4ETE which contains an RDI information signal. Otherwise,—in the case of a VC—the generator circuit **16** is instructed to generate an OAM cell of the type F5ETE which contains an RDI information signal.

If RDIN is unequal to 0, RDIN is incremented modulo-2. An OAM cell is not generated. The purpose is to maintain the time distance of 1s between two consecutive OAM cells which contain an RDI information signal.

If the conditions for generating an OAM cell are not given, RDIN is set to 0.

The same routine may be carried out in parallel for VPs and VCs with the ell types F4SEG and F5SEG. Furthermore, the extraction circuit **13** and the postprocessing ircuit **15** could process further status information to recognize other types of defects as roduced, for example, by the continuity check mechanism.

What is claimed is:

1. A transmission system comprising a plurality of network nodes for receiving and transmitting packets which contain connection-related status information signals about the transmission system, characterized in that a network node comprises at least one evaluation unit for evaluating OAM cells transmitted in an asynchronous transfer mode, said evaluation unit comprising:

means for storing status information signals extracted from a packet in a table,

means for determining, irrespective of an order of received packets, current, connection-related statuses

in a given order for each connection based upon any statuses previously stored in the table and determined thus far, as well as on the received, connection-related status information signals,

a recognition circuit for recognizing the type of OAM cell, the virtual path and, if available, the virtual connection,

an extraction circuit for extracting, at least for certain OAM cells, status information signals and transfering the status information signals to the table,

a further generator circuit, and

a postprocessing circuit including:

means for periodically extracting data relating to a connection from the table,

means for determining and storing in the table current, connection-related statuses, and

means, responsive to certain statuses, for sending a control command to said generator circuit for causing said generator circuit to generate OAM cells,

one period between OAM cell generation corresponding to the duration necessary for computing statuses of all the connections.

2. A system as claimed in claim **1**, characterized in that the evaluation unit further comprises a monitoring circuit for monitoring whether a connected line has a line defect and for passing on information about the line defect to the postprocessing circuit.

3. A system as claimed in claim **1**, characterized in that the evaluation unit further comprises:

a multiplexer for transferring either (a) cells coming from a switching network in the network node, or (b) priority cells coming from the generator circuit,

a buffer store for buffering the cells coming from the switching network,

another multiplexer for transfering either (c) cells coming from the extraction circuit or (d) priority cells coming from the generator circuit, and

a further buffer store for buffering the cells coming from the extraction circuit.

4. A system as claimed in claim **3**, characterized in that the evaluation unit further comprises a monitoring circuit for monitoring whether a connected line has a line defect and for passing on information about the line defect to the postprocessing circuit.

* * * * *

US005910803A

United States Patent [19]

Grau et al.

[11] Patent Number: 5,910,803

[45] Date of Patent: *Jun. 8, 1999

[54] NETWORK ATLAS MAPPING TOOL

[75] Inventors: **Stephen H. Grau**, Pleasanton; **Stephen Bostock**, Morgan Hill, both of Calif.

[73] Assignee: **Novell, Inc.**, Orem, Utah

[*] Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

[21] Appl. No.: **08/696,681**

[22] Filed: **Aug. 14, 1996**

[51] **Int. Cl.**[6] **G06F 3/14**; G06F 11/30

[52] **U.S. Cl.** **345/357**; 345/969; 345/349; 395/200.54

[58] **Field of Search** 345/357, 356, 345/349, 353, 969, 966, 340, 346, 335, 440, 967, 964; 707/501, 513, 514, 502; 395/200.54, 200.53

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,833,592	5/1989	Yamanaka	364/188 X
5,226,120	7/1993	Brown et al.	395/200.54
5,247,517	9/1993	Ross et al.	370/85.5
5,276,789	1/1994	Besaw et al.	345/440
5,295,244	3/1994	Dev et al.	345/357
5,394,552	2/1995	Sanchez-Frank et al.	345/349
5,412,772	5/1995	Monson	345/335
5,504,863	4/1996	Yoshida	345/356 X
5,572,640	11/1996	Schettler	345/440
5,684,967	11/1997	McKenna et al.	395/200.53 X
5,715,432	2/1998	Xu et al.	345/356 X
5,751,965	5/1998	Mayo et al.	395/200.54
5,793,974	8/1998	Messinger	395/200.54
B1 4,555,775	12/1995	Pike	345/340 X

OTHER PUBLICATIONS

Smith, Catherine J.; Kulakow, Arthur J.; Gannon, Kathleen L.; "HP Open View Windows: A User Interface for Network Management Solutions," Hewlett–Packard Journal, Apr. 1990, pp. 60–65.

AT&T News Release dated Nov. 13, 1991; "Tridom Offers Powerful New Network Management Tools".

"Management system designed for Synoptics Ethernet. (product announcement)," 1988 Fairchild Publications Inc.

"Physical layer network management: the missing link (local area networks)," 1989 Horizon House Publications Inc.

"Open View has more features than the other two products. (Open View Network Node Manager 3.1, network management software from Hewlett–Packard Co.) (Software Review) (one of three evaluations of SNMP–based remote management products in 'Compatibility in Three Consoles') (Simple Network Management Protocol) (Evaluation)," 1992 Ziff–Davis Publishing Company.

"IBM's NetView/6000 challenges the best. (Network Edition: First Looks) (Software Review) (network management software) (Evaluation)," 1993 Ziff–Davis Publishing Company.

"NetView/6000 slingshots ahead. (IBM's network management software ahead of HP's HP Open View; IBM licenses technology to DEC, plans to jointly develop new applications)," 1993 Cardinal Business Media Inc.

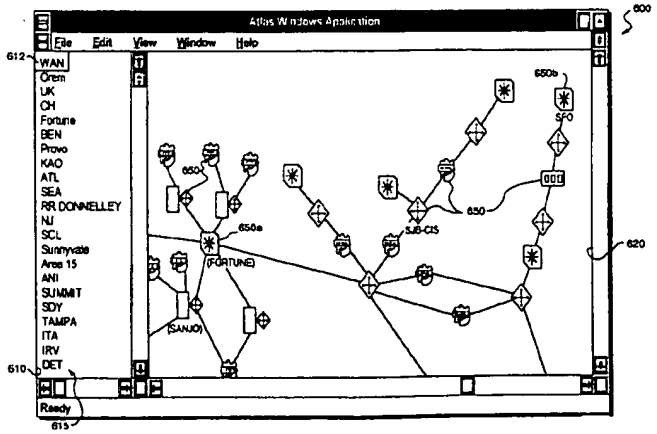
(List continued on next page.)

Primary Examiner—Raymond J. Bayerl
Attorney, Agent, or Firm—Cesari & McKenna, LLP

[57] **ABSTRACT**

A network mapping tool efficiently organizes and displays topology data of an internetwork computing system as a hierarchical collection of network maps, i.e., a network atlas. The tool includes a management server that collects, organizes and records the topology data as an atlas on a network topology database. A management console interacts with the server to provide a graphical user interface for displaying the atlas on a computer screen in a variety of views that facilitate comprehension of logical relationships between various components of the system.

11 Claims, 11 Drawing Sheets



OTHER PUBLICATIONS

"LAN/WAN mgmnt: Ungermann–Bass intros multitasking network management system; users commit to new platform. (new multitasking, microcomputer–based NetDirector handles physical management through implementation of client/server architectuer on LAN Manager–based local and wide area networks; beta customers satisfied with and migrate to new physical network management platform) (product announcement)," 1990 Edge Publishing.

"Ungermann–Bass OS/2 suite to oversee disparate LANs. (Ungermann–Bass Inc. introduces NetDirector OS/2 package for managing multivendor local area networks) (product announcement)," 1990 Ziff–Davis Publishing Company.

"Ungermann–Bass joins the personal computer–based network management fray. (NetDirector network management software) (product announcement)," 1990 Apt Data Services (UK).

"Accumaster Integrator Release 3.0 expands SNA capabilities," ATT News Release dated Jun. 17, 1992.

Muller, Nathan J., "Intelligent Hubs," 1993 Artech House, Inc., pp. 73–290.

HP Network Node Manager, http://www.dmo.hp.com/nsmd/ov/nnm.html, Network Node Manager, The Industry's Best Network Management Solution for Multivendor TCP/IP Environments Nov. 20, 1995, 2 pages, Hewlett Packard.

NetView/6000 and NetView for AIX, http://www.raleigh.ibm.com/nv6/nv6over.html, Managing A World of Difference, 1995 IBM Corporation, 3 pages, Nov. 20, 1995.

Catalog: Spectrum 3.0, http://www.ctron.com/Catalog/Net–Management/Spectrum.html, Spectrum 3.0, Detailed Graphical Views of your Network, 4 pages, Nov. 20, 1995, Product Catalog.

ManageWise 2.0, The smart way to manage your network, Novell, Intel Network Technology From Intel, 4 pages.

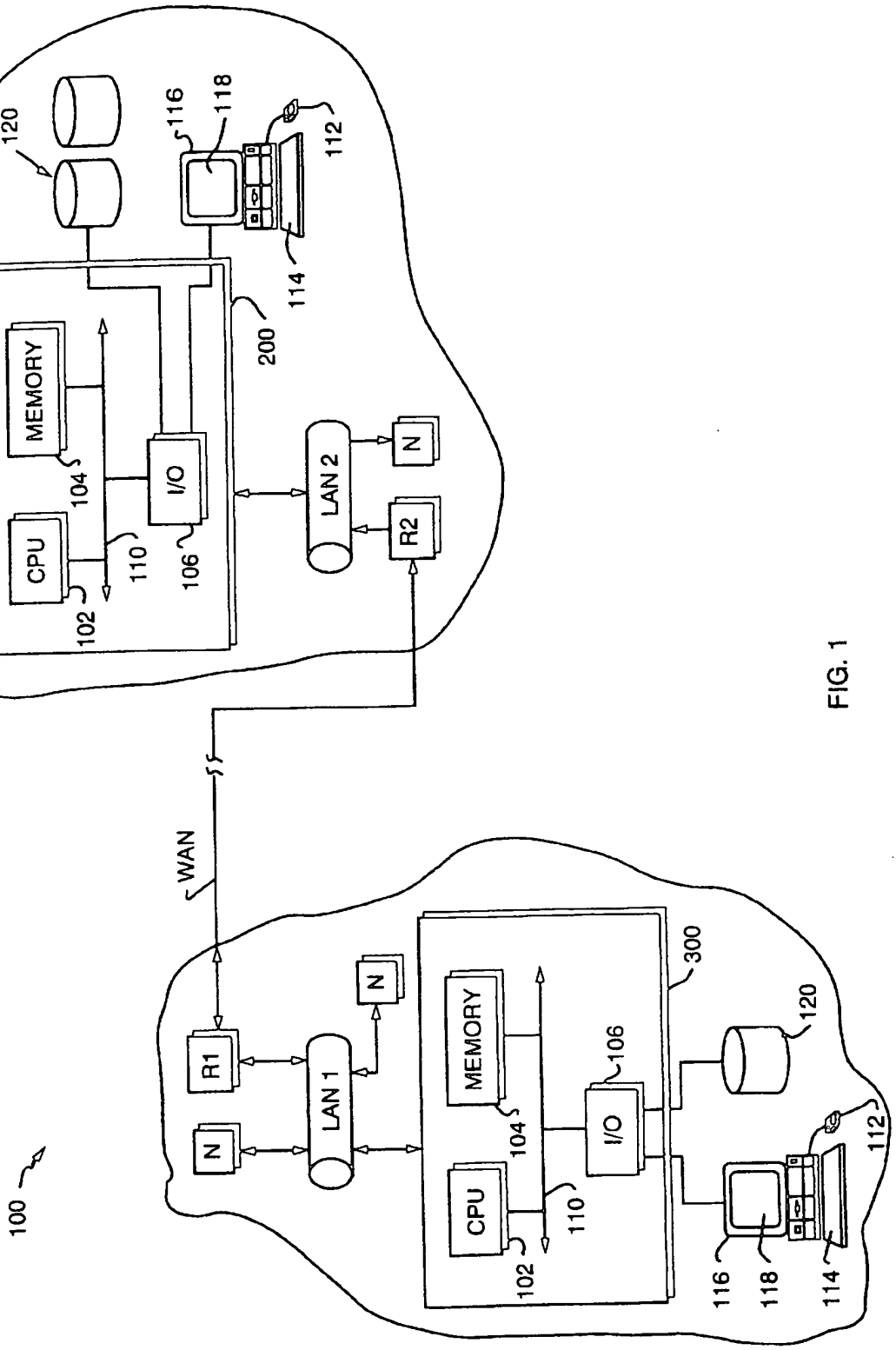

100
WAN
120
116
118
112
114
200
MEMORY
104
I/O
106
CPU
110
102
LAN 2
N
R2
R1
LAN 1
300

FIG. 1

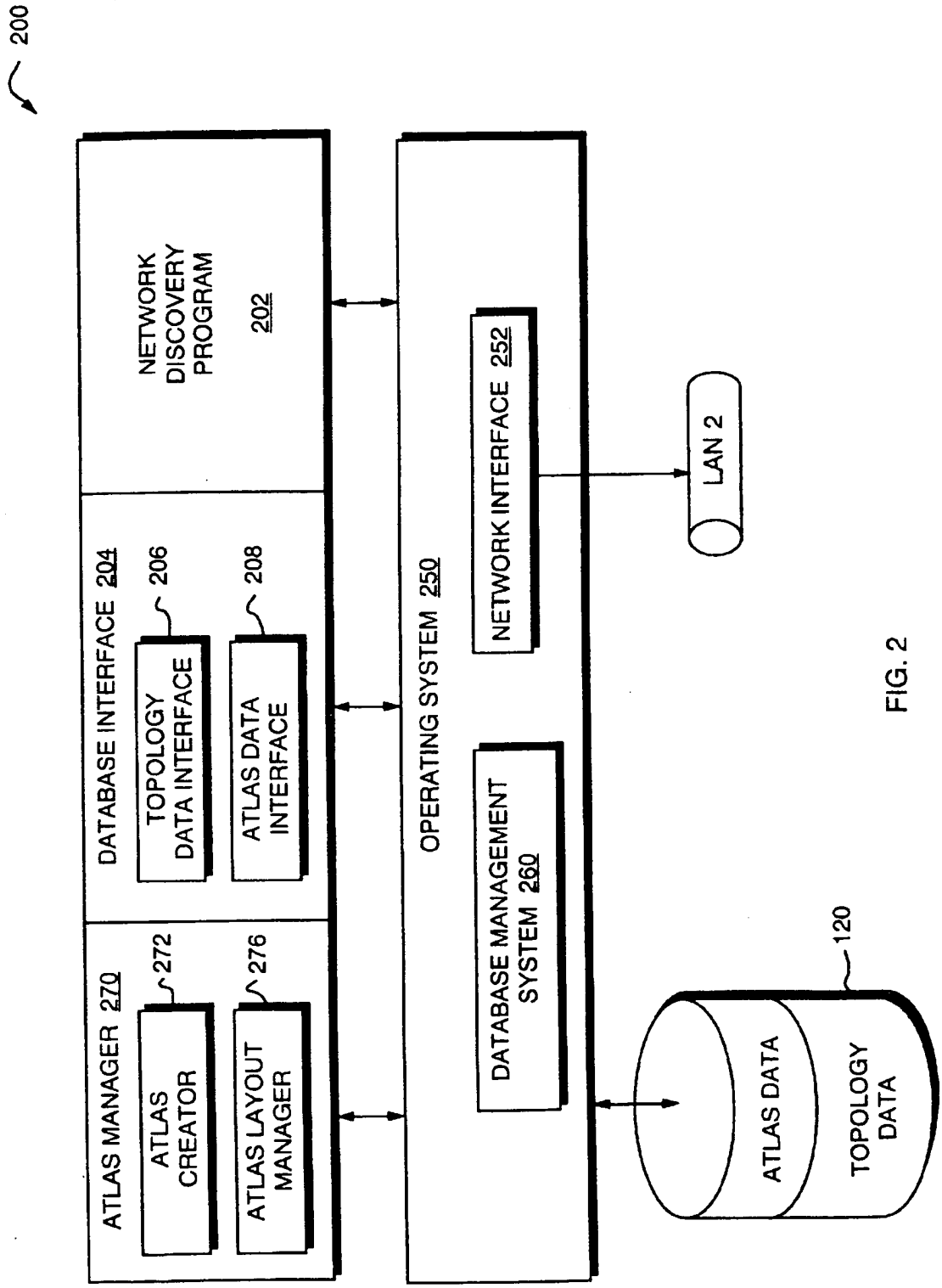

200
NETWORK DISCOVERY PROGRAM 202
DATABASE INTERFACE 204
206
TOPOLOGY DATA INTERFACE
208
ATLAS DATA INTERFACE
ATLAS MANAGER 270
272
ATLAS CREATOR
276
ATLAS LAYOUT MANAGER
OPERATING SYSTEM 250
NETWORK INTERFACE 252
LAN 2
DATABASE MANAGEMENT SYSTEM 260
120
ATLAS DATA
TOPOLOGY DATA

FIG. 2

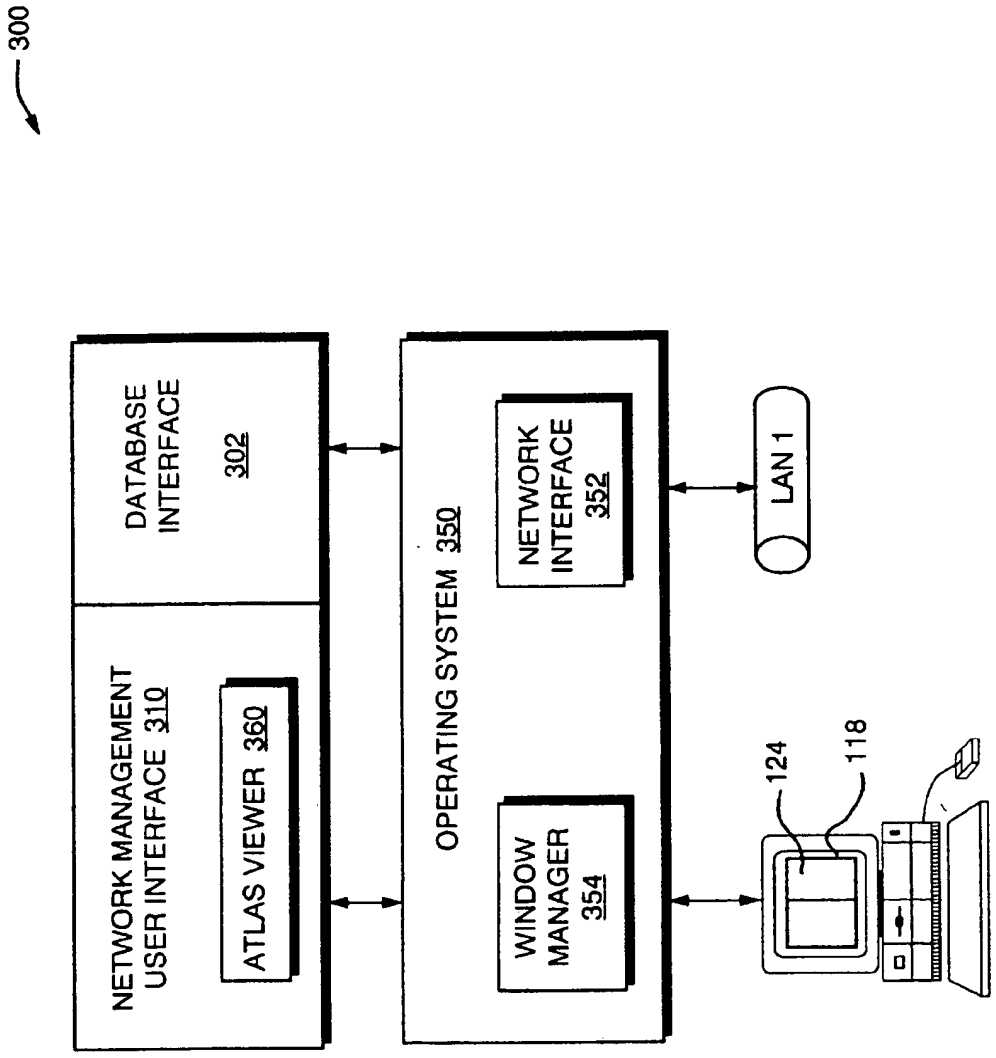

300
DATABASE INTERFACE 302
NETWORK MANAGEMENT USER INTERFACE 310
ATLAS VIEWER 360
OPERATING SYSTEM 350
NETWORK INTERFACE 352
WINDOW MANAGER 354
LAN 1
124
118

FIG. 3

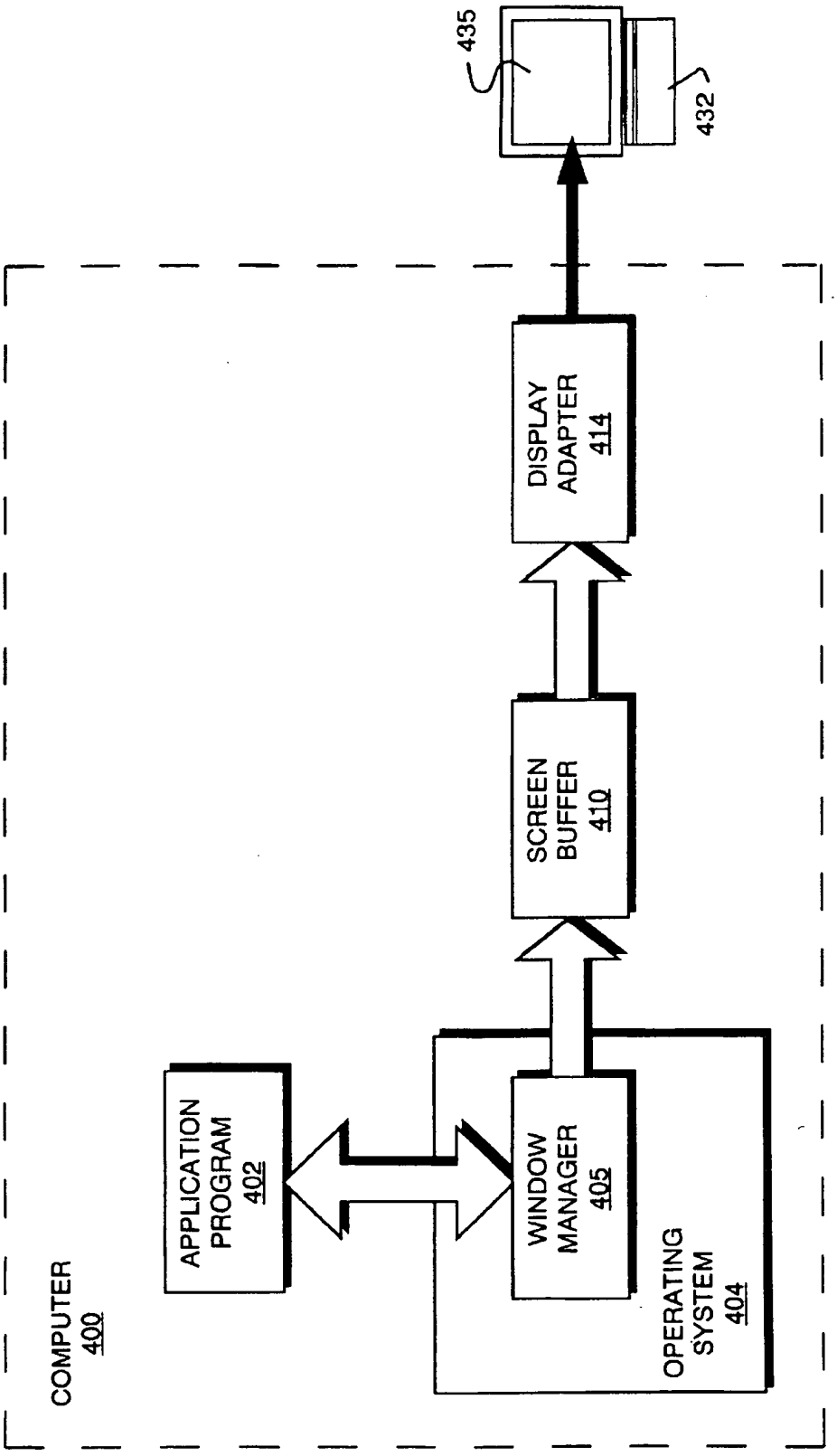

435
432
DISPLAY ADAPTER 414
SCREEN BUFFER 410
APPLICATION PROGRAM 402
WINDOW MANAGER 405
OPERATING SYSTEM 404
COMPUTER 400

FIG. 4

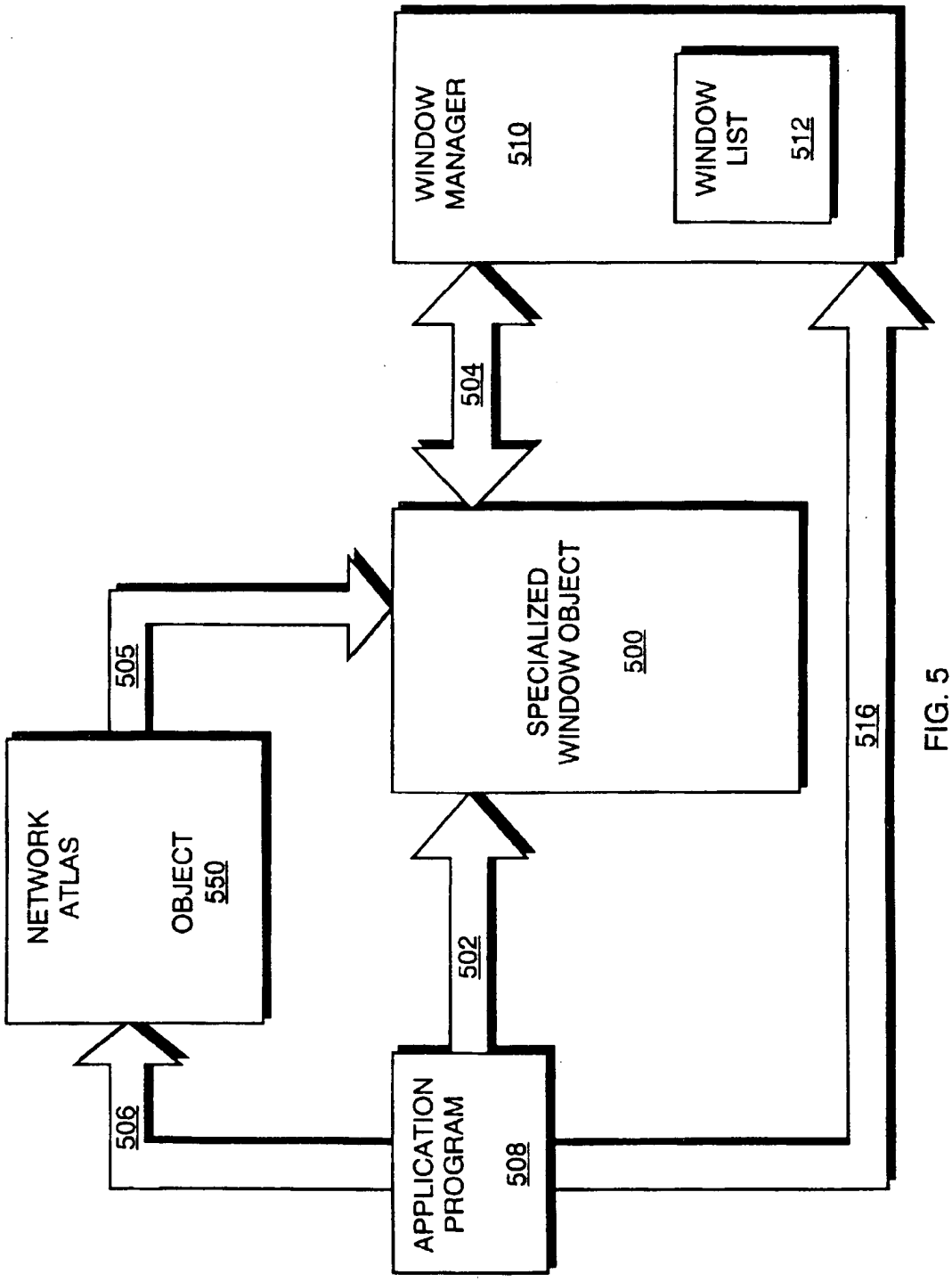

WINDOW MANAGER 510
WINDOW LIST 512
504
SPECIALIZED WINDOW OBJECT 500
505
NETWORK ATLAS OBJECT 550
502
506
APPLICATION PROGRAM 508
516

FIG. 5

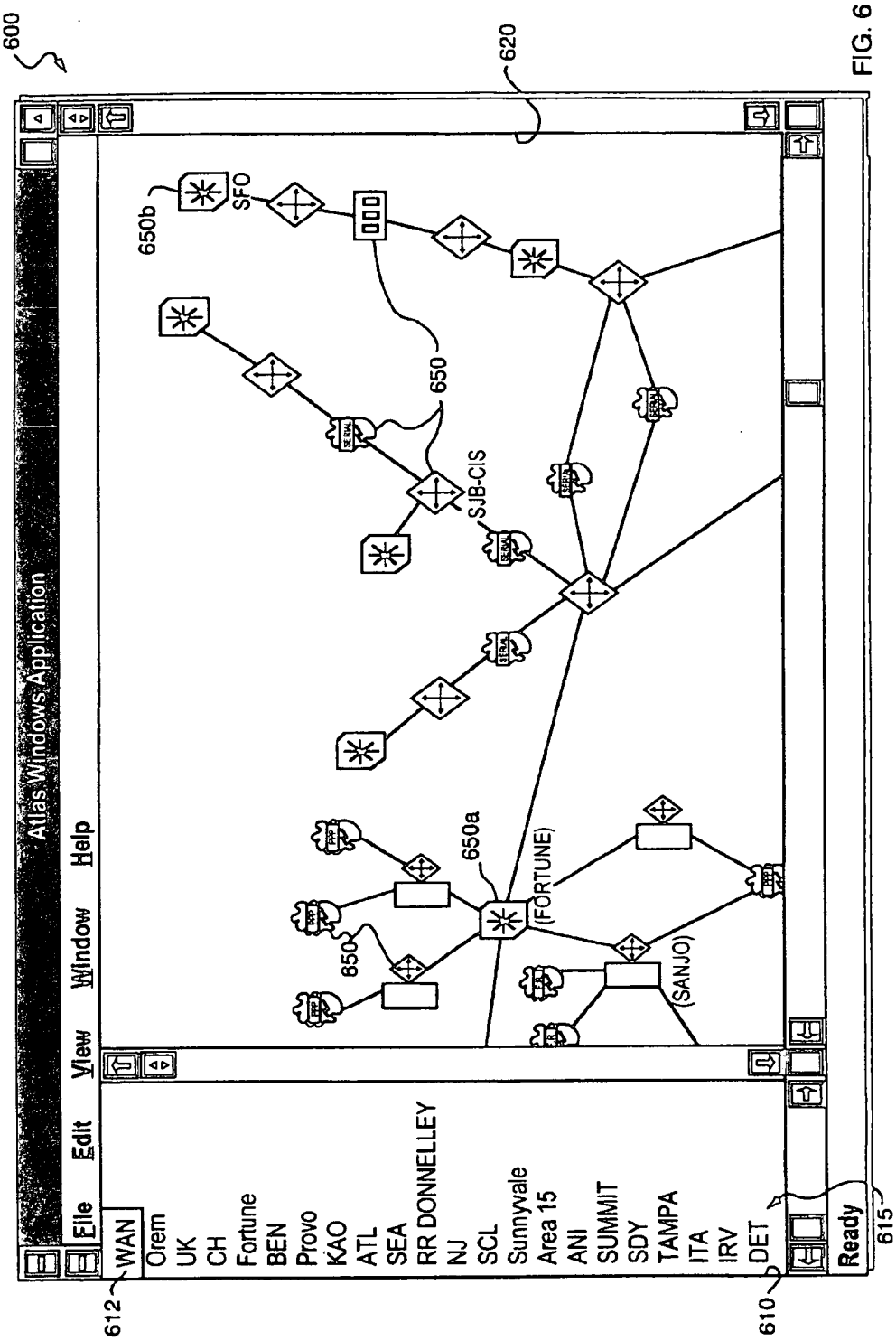

600
620
650b
SFO
650
SJB-CIS
650a
(FORTUNE)
850
(SANJO)
Atlas Windows Application
File Edit View Window Help
WAN
Orem
UK
CH
Fortune
BEN
Provo
KAO
ATL
SEA
RR DONNELLEY
NJ
SCL
Sunnyvale
Area 15
ANI
SUMMIT
SDY
TAMPA
ITA
IRV
DET
Ready
612
610
615

FIG. 6

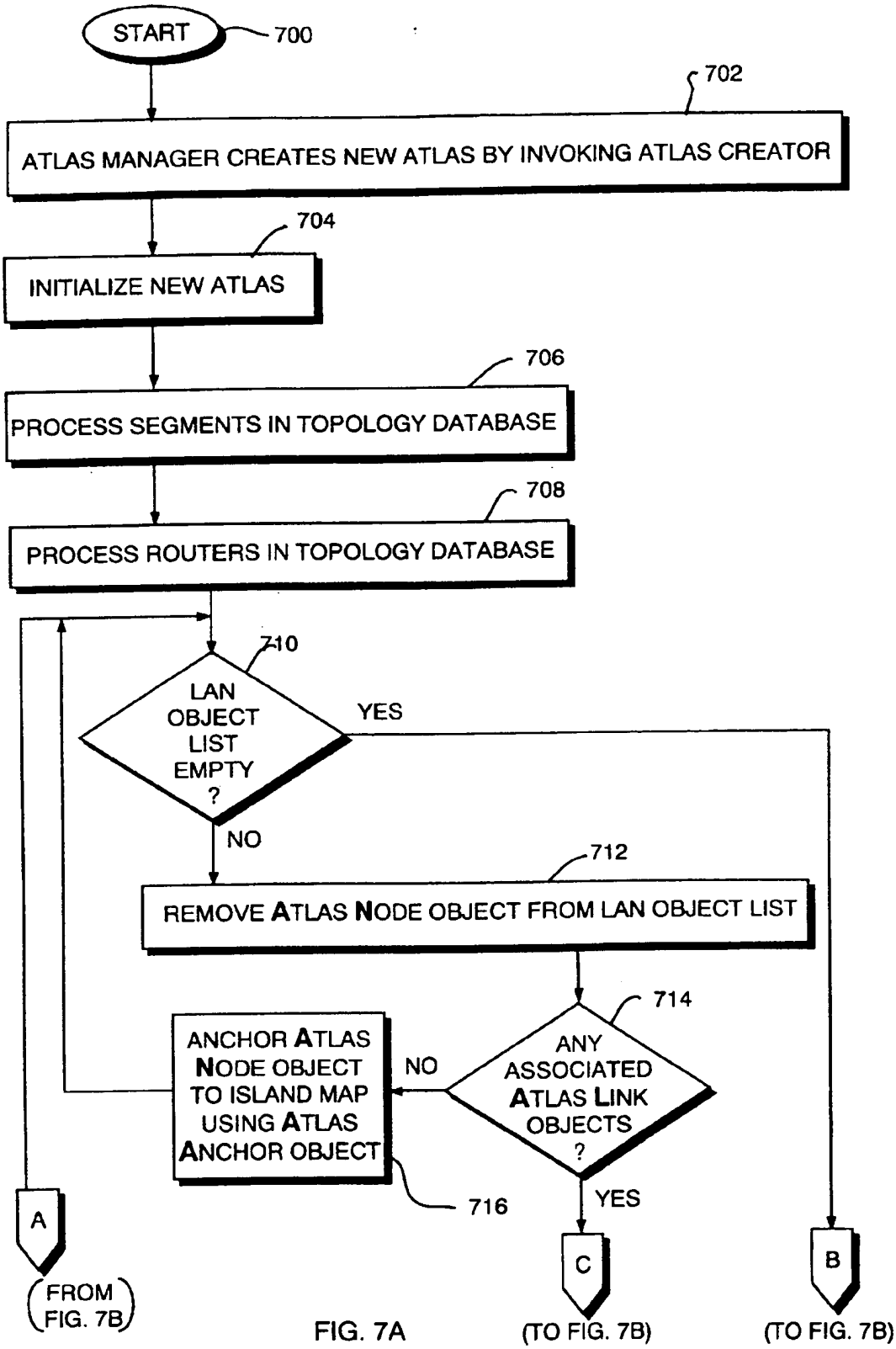

START
700
702
ATLAS MANAGER CREATES NEW ATLAS BY INVOKING ATLAS CREATOR
704
INITIALIZE NEW ATLAS
706
PROCESS SEGMENTS IN TOPOLOGY DATABASE
708
PROCESS ROUTERS IN TOPOLOGY DATABASE
710
LAN OBJECT LIST EMPTY ?
YES
NO
712
REMOVE ATLAS NODE OBJECT FROM LAN OBJECT LIST
714
ANY ASSOCIATED ATLAS LINK OBJECTS ?
NO
YES
ANCHOR ATLAS NODE OBJECT TO ISLAND MAP USING ATLAS ANCHOR OBJECT
716
A
FROM FIG. 7B
C
(TO FIG. 7B)
B
(TO FIG. 7B)

FIG. 7A

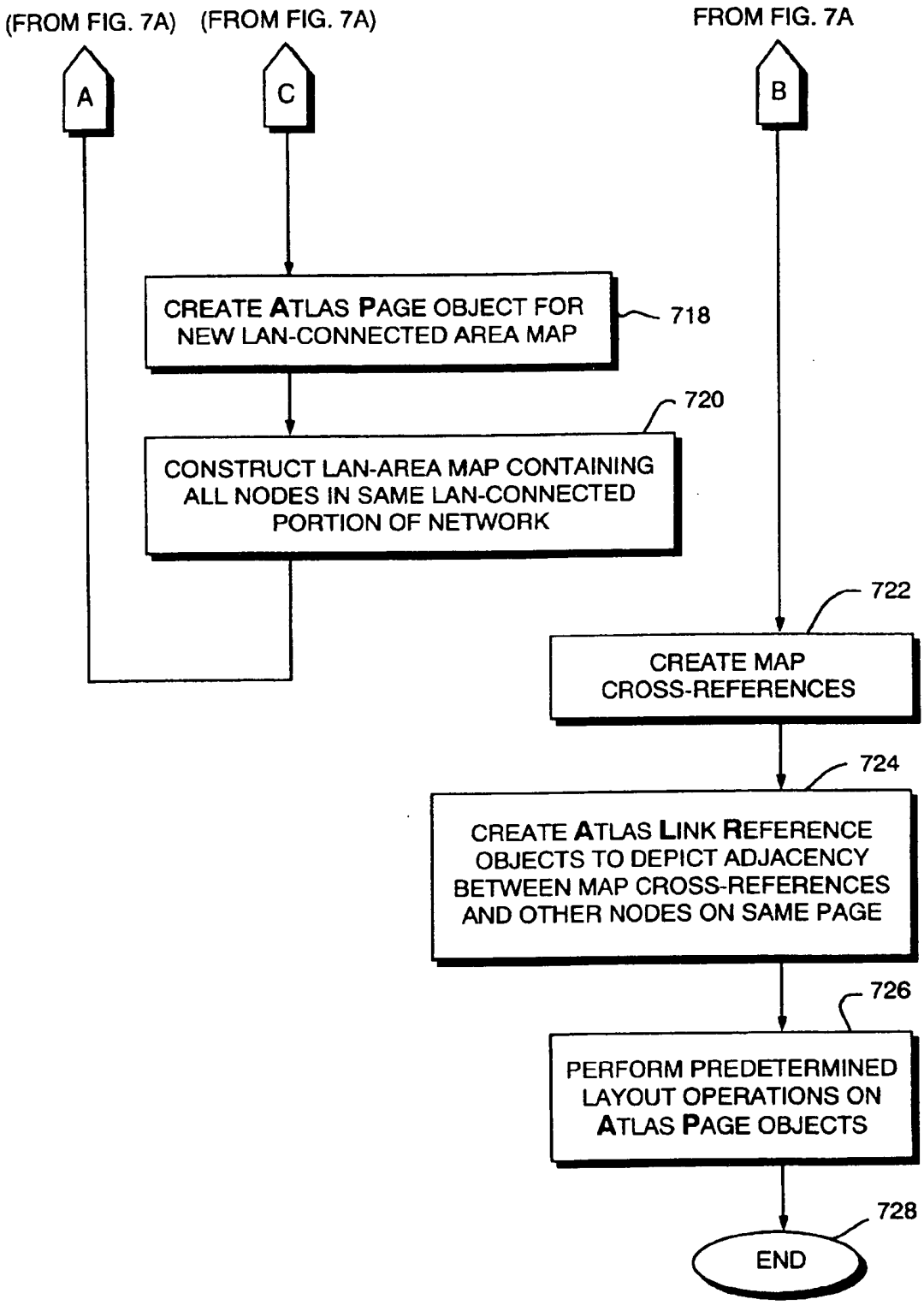

(FROM FIG. 7A)
(FROM FIG. 7A)
FROM FIG. 7A
A
C
B
CREATE ATLAS PAGE OBJECT FOR NEW LAN-CONNECTED AREA MAP
718
720
CONSTRUCT LAN-AREA MAP CONTAINING ALL NODES IN SAME LAN-CONNECTED PORTION OF NETWORK
722
CREATE MAP CROSS-REFERENCES
724
CREATE ATLAS LINK REFERENCE OBJECTS TO DEPICT ADJACENCY BETWEEN MAP CROSS-REFERENCES AND OTHER NODES ON SAME PAGE
726
PERFORM PREDETERMINED LAYOUT OPERATIONS ON ATLAS PAGE OBJECTS
728
END

FIG. 7B

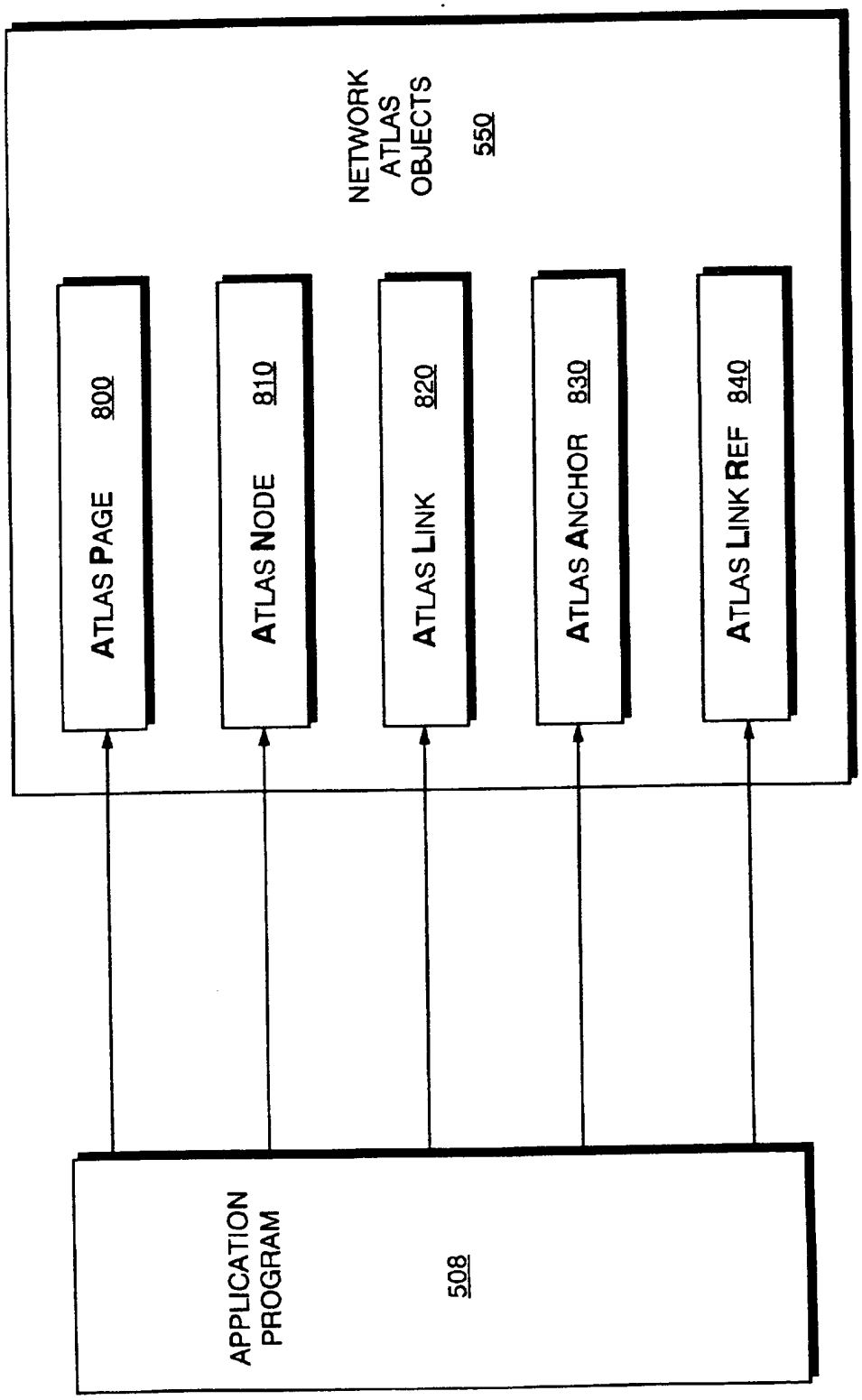

NETWORK ATLAS OBJECTS 550
ATLAS PAGE 800
ATLAS NODE 810
ATLAS LINK 820
ATLAS ANCHOR 830
ATLAS LINK REF 840
APPLICATION PROGRAM 508

FIG. 8

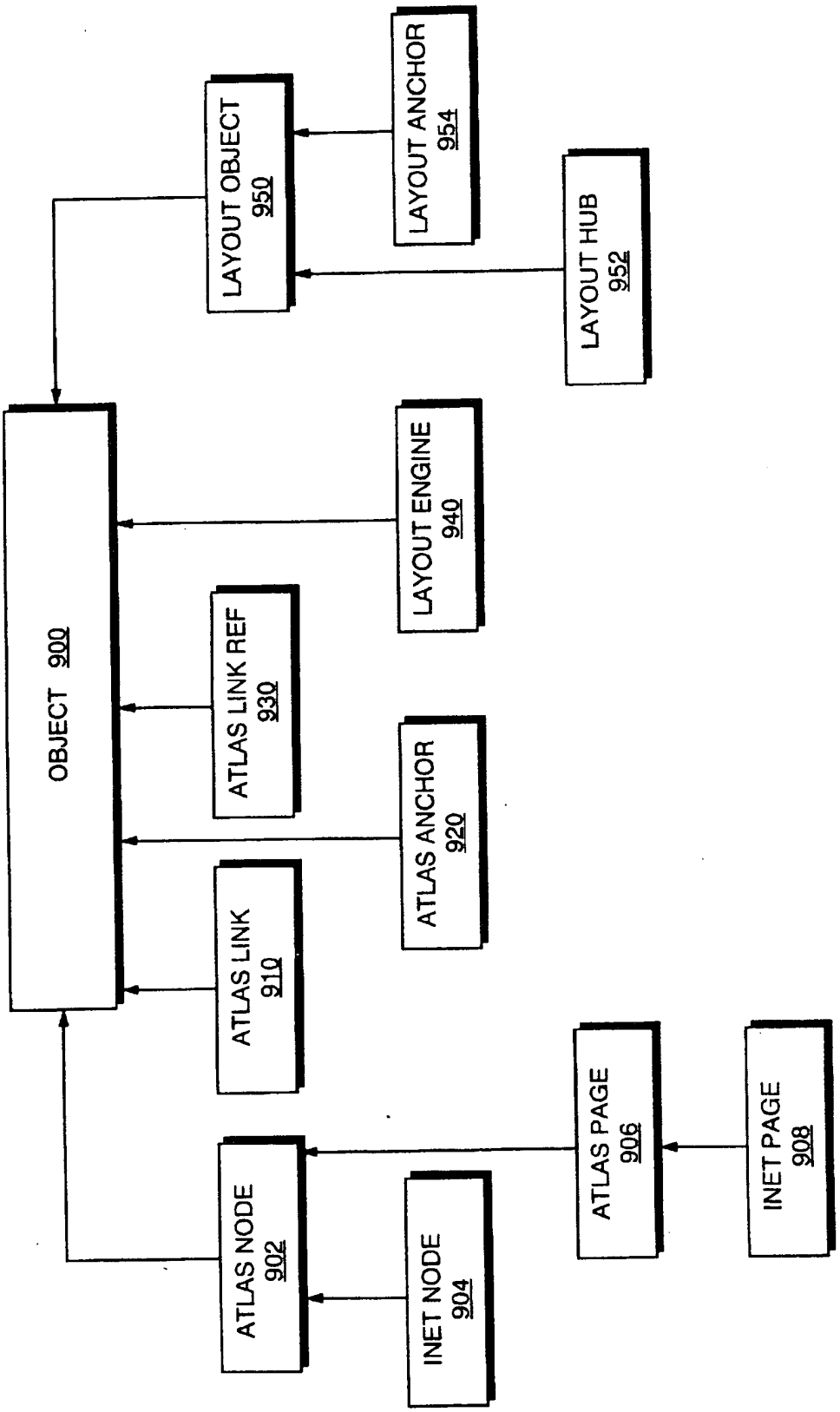

OBJECT 900
LAYOUT OBJECT 950
LAYOUT ANCHOR 954
LAYOUT HUB 952
LAYOUT ENGINE 940
ATLAS LINK REF 930
ATLAS ANCHOR 920
ATLAS LINK 910
ATLAS NODE 902
INET NODE 904
ATLAS PAGE 906
INET PAGE 908

FIG. 9

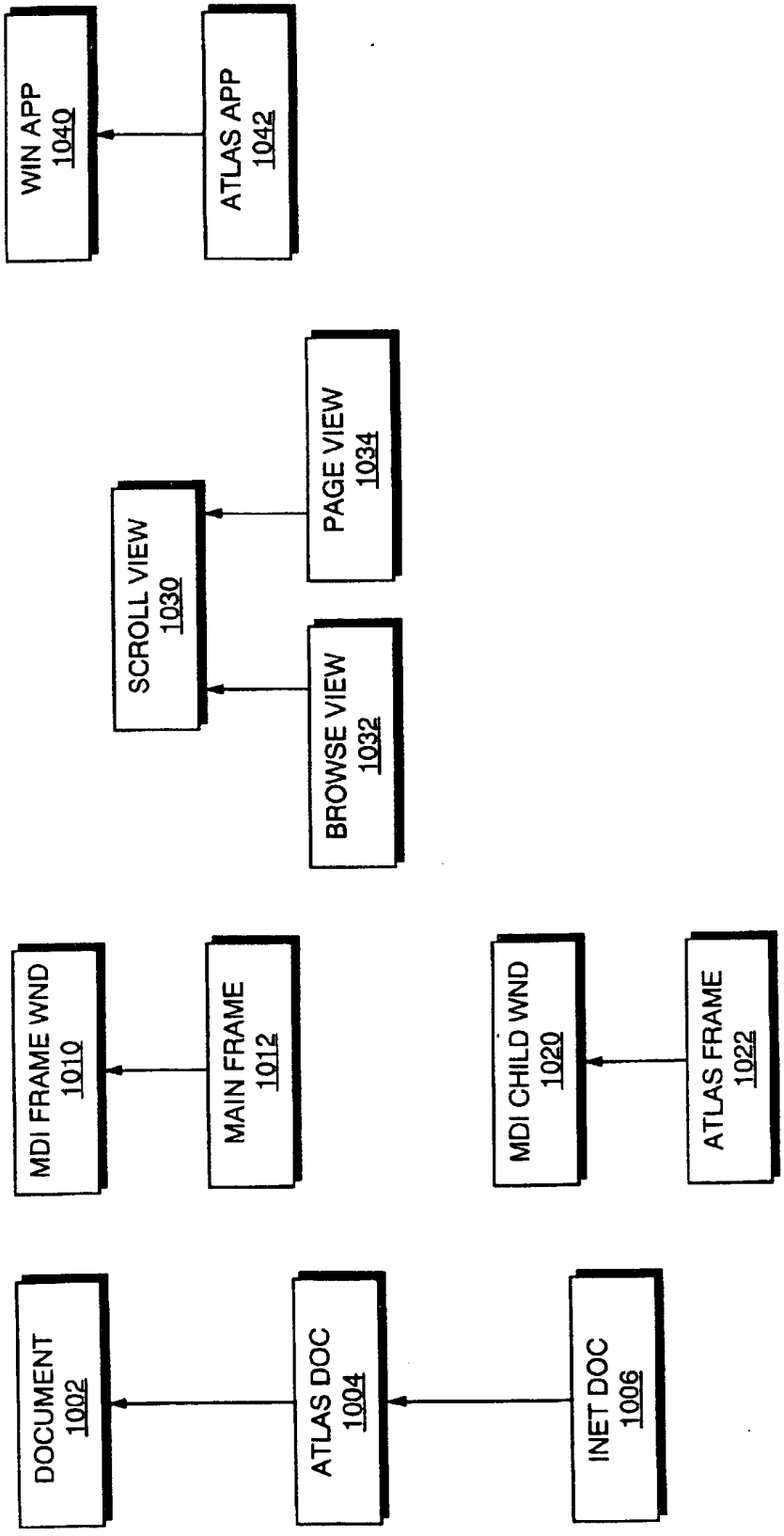

WIN APP
1040
ATLAS APP
1042
PAGE VIEW
1034
SCROLL VIEW
1030
BROWSE VIEW
1032
MDI FRAME WND
1010
MAIN FRAME
1012
MDI CHILD WND
1020
ATLAS FRAME
1022
DOCUMENT
1002
ATLAS DOC
1004
INET DOC
1006

FIG. 10

NETWORK ATLAS MAPPING TOOL

CROSS-REFERENCE TO RELATED APPLICATION

This invention is related to copending and commonly assigned U.S. patent application Ser. No. 08/698,057, titled METHOD AND APPARATUS FOR ORGANIZING OBJECTS OF A NETWORK MAP, which application was filed on even date herewith.

FIELD OF THE INVENTION

This invention relates generally to internetwork computing systems and, more specifically, to an improved network mapping tool for efficiently managing complex internetwork computing systems.

BACKGROUND OF THE INVENTION

An internetwork computing system is a geographically distributed collection of interconnected network segments for transporting data between computing nodes, such as routers. The network segments are typically local area networks (LANs) coupled together by point-to-point wide area networks (WANs). A LAN is a limited area network that typically consists of a transmission medium, such as coaxial cable or twisted pair, for connecting the nodes, while a WAN may be a public or private telecommunications facility that interconnects widely dispersed LANs. Collectively, the LAN and WAN segments may be configured to form a complex topology of internetworked computing nodes that interact according to a predefined set of rules or protocols.

A network mapping system is used to manage such complex internetwork computing systems by providing network management service tools. These services may be implemented in accordance with a client/server architecture, wherein the clients, e.g., personal computers or workstations, are responsible for interacting with the users and the servers are computers configured to perform the services as directed by the clients. Furthermore, the service tools may range from map-drawing software to capabilities for automatically discovering the topology of network segments using a variety of LAN, WAN and protocol technologies. Information pertaining to the discovered topology is typically gathered to build a coherent database model of the internetwork system.

However, the tools used for viewing the topology of the network are typically difficult to comprehend primarily because the entire internetwork is often presented as a single, monolithic "page" on a computer display. As a result, the layout of the internetwork is such that nodes typically appear far away from their neighboring nodes, making it hard to discern relationships among elements of the system. Furthermore, the internetwork layout typically changes as new nodes are discovered.

SUMMARY OF THE INVENTION

The present invention relates to a network mapping tool for efficiently organizing and displaying topology data of an internetwork computing system as a hierarchical collection of network maps, i.e., a network atlas. In accordance with the invention, the mapping tool includes a management server that collects, organizes and records the topology data as the atlas on a network topology database. A management console interacts with the server to provide a graphical user interface for displaying the atlas on a computer screen in a variety of views that facilitate comprehension of logical relationships between various components of the system.

Specifically, the management server comprises an atlas manager that coordinates access to the atlas database for editing and viewing the atlas maps. The atlas manager, in turn, comprises an atlas creator that creates the atlas from the topology data and a layout manager for executing layout operations that enable display of the created atlas maps in a manner that is visually appealing to a user and easier to comprehend.

On the other hand, the management console preferably comprises an atlas viewer for displaying selected maps of the atlas that are visible through specialized windows of the graphical user interface. To that end, the atlas viewer cooperates with the atlas manager to retrieve the topology data describing the structure and layout of the selected maps within the network atlas. Maps are preferably selected using a browser navigation facility which allows a user to easily switch between maps of interest by selecting the name of a desired map.

For example, as the user "browses" the atlas, the viewer module fetches the appropriate data from the network topology database needed to execute navigation and display functions. These functions are preferably implemented in response to user commands provided from an input device, such as a mouse. As a result of these commands, the atlas viewer navigates, i.e., switches, from a currently displayed map and scrolls to an area of interest.

Similarly, the atlas viewer responds to user commands from the input device to configure the atlas in a customized arrangement specified by the user; alternatively, the atlas manager may create a default configuration of the atlas. As noted, the atlas viewer interacts with the atlas manager to record any customized changes to the default arrangement in the atlas database.

In the illustrative embodiment, the network mapping tool is preferably embodied as a customized framework having generic base classes for defining novel window and network atlas objects. The network atlas objects generally represent a connected graph of linked nodes within the hierarchical atlas of maps manifested on specialized "viewer" windows defined by the window objects. In particular, the nodes of the maps are router and segment objects, while the "links" are preferably interfaces between those objects.

Each map of the atlas depicts a portion or page of the internetwork system, with a top-level page illustrating the overall WAN structure of the system and subordinate-level pages showing the structures of specific campuses, i.e., LAN-connected areas of the system. Notably, each map may refer to other maps of the atlas by way of map cross-reference objects. The router, segment and map reference objects are preferably displayed as icons at various locations on a map, with the links shown as lines coupling the icons.

BRIEF DESCRIPTION OF THE DRAWINGS

The above and further advantages of the invention may be better understood by referring to the following description in conjunction with the accompanying drawings in which like reference numerals indicate identical or functionally similar elements:

FIG. 1 is a block diagram of an internetwork computing system including a collection of network segments connected to a plurality of stations;

FIG. 2 is a highly schematized diagram of the software components of a management server station coupled to the internetwork computing system of FIG. 1;

FIG. 3 is a highly schematized diagram of the software components of a management console station coupled to the internetwork computing system of FIG. 1;

FIG. 4 is a schematic diagram illustrating the interaction of an application program and an operating system of a computer 400, which is similar to the management console station of FIG. 1;

FIG. 5 shows the interaction between an application program and a window manager to create and manipulate novel network atlas objects in accordance with the invention;

FIG. 6 is a diagram of a specialized viewer window in accordance with the invention;

FIGS. 7A and 7B are flowcharts illustrating the sequence of steps comprising a novel atlas creation process in accordance with the invention;

FIG. 8 is a schematic diagram of the network atlas objects used by the application program in accordance with the invention;

FIG. 9 is a simplified class hierarchy diagram for the network atlas objects of FIG. 8; and

FIG. 10 shows simplified class hierarchy diagrams for various application, document and window objects of the invention.

DETAILED DESCRIPTION OF ILLUSTRATIVE EMBODIMENT

FIG. 1 is a block diagram of an internetwork computing system 100 comprising a collection of network segments connected to a plurality of stations. The nodes are typically general-purpose computers comprising a management server station 200, a management console station 300, a plurality of end stations N and a plurality of intermediate stations R1–R2. Each station typically comprises a central processing unit (CPU) 102, a memory unit 104 and an input/output (I/O) unit 106 interconnected by a system bus 110. The memory unit 104 may comprise storage locations typically composed of random access memory (RAM) devices, which are addressable by the CPU 102 and I/O unit 106. An operating system, portions of which are typically resident in memory and executed by CPU, functionally organizes the station by, inter alia, invoking network operations in support of application programs executing in the CPU.

The I/O unit 106, in turn, connects the station to the network segments, a conventional display monitor 116 and a mass storage device, such as disk 120. The display monitor 116 includes a display screen 118 and cursor control devices, such as a mouse 112 and keyboard 114. In the case of the management server 200, the disk may function as a network topology database 120 for storing topology data relating to the system 100, as described further herein. Typically, the I/O unit 106 receives information, such as control and data signals, from the mouse or keyboard and provides that information to the CPU 102 for transfer over the network segments, for storage on the database or for display on the screen 118.

The network segments included within system 100 are preferably local area networks (LANs) 1–2 coupled to a wide area network (WAN) by intermediate stations R1–R2. The intermediate stations R1–R2 may be routers configured to facilitate the flow of data throughout the system 100 by routing that data to the proper receiving stations. Collectively, the LAN and WAN segments may be configured to form a complex topology of internetworked computing stations that interact according to predefined set of protocols.

As noted, a network mapping system is typically used to manage such a complex internetwork computing system by providing network management services. According to the invention, a novel network mapping tool is provided for implementing these services in accordance with a client/server architecture, wherein the client is the management console node 300 that is responsible for interacting with a user and the management server node 200 functions as the server to perform services as directed by the client. As described herein, this tool organizes and displays topology data as a hierarchical collection of network maps, i.e., a network atlas. Each map of the atlas shows a portion of the computing system as a connected graph of nodes; these nodes generally comprise the router and network segments.

FIG. 2 is a highly schematized diagram of the software components of the management server station 200. The management server 200 generally monitors the internetwork in order to collect, organize and record topology data and atlas data in the topology database 120. To that end, the server includes a database interface component 204 having a topology data interface 206 and an atlas data interface 208 for accessing the data in the database. In addition, a conventional network discovery component 202 is provided for automatically discovering the topology of network segments using a variety of LAN, WAN and protocol technologies.

The server further includes operating system software 250 having a collection of utility programs for controlling the operation of the station 200. These utility programs generally include a network interface 252 that provides the server access to the network system 100 and a database management system 260 that communicates with the interfaces 206 and 208 when exchanging data with the database 120.

In accordance with an aspect of the invention, the management server 200 also comprises an atlas manager 270 that coordinates access to the database 120 for editing and viewing the atlas maps. The atlas manager, in turn, comprises an atlas creator 272 that creates the atlas from the topology data and a layout manager 276 for executing layout operations that enable display of the created atlas maps in a manner that is visually appealing to a user. An example of these layout operations is provided in copending and commonly assigned U.S. patent application Ser. No. 08/698,057 titled Method and Apparatus for Organizing Objects of a Network Map, filed on even date herewith, which application is incorporated by reference as though fully set forth herein.

FIG. 3 is a highly schematized diagram of the software components of the management console station 300. These components generally include an operating system 350 which interacts with various application program components to provide high-level functionality, including a direct interface with the user. The application program components include a database interface component 302 configured to interact with the management server 200 when manipulating data and controlling the operations of that server. In addition, a network management user interface component 310 allows the user to view aspects of the internetwork system (e.g., configuration, state and history) and perform management-type operations on the internetwork.

Utility programs such as a network interface 352 are included in the operating system software to provide the console with access to the network system 100 and a window manager 354. The window manager is a system software routine that is generally responsible for managing windows that the user views during operation of an application program. That is, it is generally the task of the window manager to keep track of the locations and sizes of the windows and window areas which must be drawn and redrawn in connection with such application execution.

Broadly stated, the application programs make use of the functions of the utility programs by issuing a series of task commands to the operating system which then performs the requested task. For example, the database interface component **302** may request that the network interface **352** initiate transfer of information over LAN 1; likewise, the network management user interface **310** may request that the window manger **354** display certain information on a window of the screen **118** for presentation to the user.

Further to this latter example, the console **300** also comprises an atlas viewer application component **360** which, in connection with the window manager **354**, provides a graphical user interface for displaying the topology data as an atlas on specialized viewer windows **124** of the display screen **118**. Specifically, the atlas viewer and window manager operate to display views of selected maps of the atlas on the viewer windows **124** to facilitate comprehension of logical relationships between various components of the system **100**. The atlas viewer **360** also cooperates with the atlas manager **270** to retrieve the topology data describing the structure and layout of the selected maps within the network atlas. More specifically, as the user "browses" the atlas, the viewer module fetches the appropriate data from the network topology database needed to execute the navigation and display functions.

Each station of the system **100** may be a general-purpose computer, such as a personal computer of the IBM® series of computers sold by International Business Machines Corp., although the invention may also be practiced in the context of other types of computers. These computers have resident thereon, and are controlled and coordinated by, operating system software, such as the IBM OS/2® operating system or the Microsoft® Windows® operating system. In addition, a window environment, such as the Windows® graphical user interface, is preferably displayed on the screen **118** as a graphical display to facilitate interactions between the user and the station. The graphical display is typically arranged to resemble a desktop and, as described herein, the application programs execute in the novel viewer windows **124** of the screen **118**.

The invention herein features, along with these specialized windows, the provision of the new network mapping tool which, when invoked, cause actions to take place that enhance the ability of a user to interact with the computing system. As noted, the novel mapping tool efficiently organizes and displays the topology data of the system as novel atlas elements. This new behavior of the system is brought about by the interaction of the mapping tool with a series of system software routines associated with the operating system. These system routines, in turn, interact with the application program components to create the viewer windows, and atlas elements, as described herein.

FIG. **4** is a schematic illustration of the interaction of an application program **402** and an operating system **404** of a computer **400**, which is similar to, and has equivalent elements of, the console station **300**. The application program **402** and the operating system **404** interact to control and coordinate the operations of the computer **400**. In order to display information on a screen display **435**, application program **402** generates and sends display commands to a window manager **405** of the operating system **404**. The window manager program **405** stores information directly into a screen buffer **410**. Under control of various hardware and software in the system, the contents of the screen buffer **410** are read out of the buffer and provided to a display adapter **414**. The display adapter contains hardware and software (sometimes in the form of firmware) which converts the information in the screen buffer **410** to a form which can be used to drive a display screen **435** of a monitor **432**.

In a preferred embodiment, the invention described herein is implemented in a C++ programming language, such as Microsoft Visual C++, using object-oriented programming (OOP) techniques. The C++ language is well-known and many articles and texts are available which describe the language in detail. In addition, C compilers are commercially available from several vendors. Accordingly, for reasons of clarity, the details of the C++ language and the operation of its compiler will not be further discussed.

As will be understood by those skilled in the art, OOP techniques involve the definition, creation, use and destruction of "objects". These objects are software entities comprising data elements and routines, or functions, which manipulate the data elements. The data and related functions are treated by the software as an entity that can be created, used and deleted as if it were a single item.

Objects are defined by creating "classes" which are not objects themselves, but which act as templates that instruct the compiler how to construct an actual object. A class may, for example, specify the number and type of data variables and the steps involved in the functions which manipulate the data. Objects are created and destroyed at run-time according to functions defined by the classes; the functions are compiled into executable statements by the compiler. Objects may be used by manipulating their data and invoking their functions.

The principle benefits of OOP techniques arise out of three basic principles: encapsulation, polymorphism and inheritance. Specifically, objects can be designed to hide, or encapsulate, all, or a portion of, their internal data structure and internal functions. Polymorphism is a concept which allows objects and functions that have the same overall format, but that work with different data, to function differently in order to produce consistent results. Inheritance, on the other hand, allows program developers to easily reuse pre-existing programs and to avoid creating software from scratch.

In accordance with the present invention, the windows and novel atlas elements are "objects" created by the application program to communicate with the window manager. Interaction between an application program and a window manager is illustrated in greater detail in FIG. **5**. In general, an application program **508** interfaces with the window manager **510** by creating and manipulating objects. The window manager itself may be an object which is created when the operating system is started. Specifically, the application program creates specialized window objects **500**, as depicted by arrow **502**, that cause the window manager to create associated viewer windows on the display screen. In addition, the application program **508** creates individual network atlas objects **550**, indicated by arrow **506**, that are stored in each window object **500** via arrow **505**.

Since many atlas objects may be created in order to display many atlas maps on the display screen, the window object **500** communicates with the window manager **510** by means of a sequence of drawing commands issued from the window object **500** to the window manager **510**, as illustrated by arrow **504**. The application **508** also communicates with the window manager **510** by sending commands to the manager **510**, as indicated by arrow **516**. The window manager **510** maintains a window list **512** that contains each window currently in the system.

Although OOP offers significant improvements over other programming concepts, program development still requires

significant outlays of time and effort, especially if no pre-existing software programs are available for modification. Consequently, a prior art approach has been to provide a program developer with a set of pre-defined, interconnected classes which create a set of objects and additional miscellaneous routines that are all directed to performing commonly-encountered tasks in a particular environment. Such pre-defined classes and libraries are typically called "application frameworks" and essentially provide a pre-fabricated structure for a working application.

There are many kinds of application frameworks available, depending on the level of the system involved and the kind of problem to be solved. The types of frameworks range from high-level application frameworks that assist in developing a user interface, to lower-level frameworks that provide basic system software services such as communications, printing, file systems support, graphics, etc. Commercial examples of application frameworks include MacApp (Apple), Bedrock (Symantec), OWL (Borland), NeXT Step App Kit (NeXT), Smalltalk-80 MVC (ParcPlace), Java (Sun Microsystems) and, as described further herein, Microsoft Foundation Classes (MFC).

A preferred embodiment takes the concept of frameworks and applies it throughout the entire system, including the application and the operating system. For the commercial or corporate developer, systems integrator, or OEM, this means all of the advantages that have been illustrated for a framework, such as MFC, can be leveraged not only at the system level for such services as printing, graphics, multimedia, file systems and I/O operations, but also at the application level, for things such as text, graphical user interfaces and, as described herein, network mapping tools.

Referring again to FIG. **5**, the window objects **500** are elements of an improved network mapping tool having a customizable framework for greatly enhancing the ability of a user to interact with an application executing on the internetwork computing system. As described herein, the application executes in specialized viewer windows represented by the window objects. In addition, the customizable framework facilitates creation of different objects, such as the novel network atlas objects **550**, stored in the memory **104**. These objects generally represent a connected graph of linked nodes within the hierarchical atlas of maps displayed on screen **118** of monitor **116**. It should be noted that the network atlas objects may be expanded to include certain key nodes, such as servers, in the context of the internetwork maps.

FIG. **6** is a diagram of a specialized viewer window **600**, which is similar to the window **124** appearing on display screen **118**. The viewer window **600** is configured to simplify moving among locations of the atlas maps and, as such, provides navigation features within a plurality of panes **610** and **620**. In the left pane **610**, a browser **615** is provided to display a list of map names, as the right pane **620** displays a map that is selected from among those names of the browser list. Specifically, the browser navigation facility allows a user to easily switch between maps of interest by selecting the name of a desired map. The desired map may be displayed by positioning the mouse pointer and clicking on that name. The currently displayed map page is indicated by a box **612** surrounding the corresponding map name. It should be understood that other types of browsers, e.g., a tree browser depicting a directory of maps, may be used in place of the browser list. In this case, the tree browser would allow the user to go directly to a specific map from the map directory.

As noted, the atlas manager **270** (FIG. **2**) of the novel mapping tool initially (e.g., as a default) constructs a set of hierarchical, internetwork maps from the topology data stored on network topology database **120** in accordance with a novel atlas creation process. Like a traditional atlas, each map depicts a portion or page of the internetwork system on the pane **620**. A top-level page illustrates the overall WAN structure of the system by setting forth the interconnectivity between campuses, i.e., LAN-connected areas of the system. Individual LAN maps, each depicting the organizational structure of a campus, are presented on subordinate-level pages.

FIGS. **7**A and **7**B are flowcharts illustrating the sequence of steps comprising the atlas creation process. The novel process starts at Step **700** and proceeds to Step **702** where the atlas manager creates a new atlas by invoking the atlas creator from the network topology database; invocation preferably occurs in response to the user executing a File→New command from the window menu. The atlas generally comprises a 2-tier hierarchy with at least one WAN map residing at the top level of the hierarchy and a series of LAN-connected area maps arranged at a lower, subordinate level. The WAN-level map is configured to display WAN segments and routers on a created WAN page; these segments and routers interconnect various LAN-connected areas of the network. Each lower-level LAN map contains a single contiguous LAN-connected area. Cross-references for each of the LAN maps are placed on the WAN map by anchoring the LAN maps to the WAN map.

In Step **704**, the atlas creator "initializes" the new atlas by creating the novel network atlas objects described herein. For example, AtlasPage objects are created for each WAN map and Island map. An Island map is a special map that serves as a convenient way to assemble segments without interfaces to routers. These segments typically surface when the network discovery program is unable to obtain information about the interfaces between segments and routers, thus leaving the topology database in an incomplete state. The atlas creator further initializes the new atlas by creating lists such as a Segment list, LAN Object list and LAN Area list.

Preferably, the segments are classified based on each segment's data-link protocol type stored in the topology database by the network discovery component **202** or the user. For example, if the data-link protocol indicates the link is a point-to-point link, rather than a multi-access link, then the segment is classified as a WAN segment. Data-link protocols such as X.25, PPP, ATM, ISDN, Frame-Relay, T1 and T3 are examples of protocols for classifying point-to-point links as WAN segments. If the data-link protocol is not known, the segment is considered a WAN segment if it interfaces to exactly two router nodes.

In Step **706**, the atlas creator processes the segments in the topology database by enumerating the segments and creating an AtlasNode object for each enumerated segment. In particular, the AtlasNode objects are added to the Segment list and each associated segment is classified (marked) as a WAN or LAN segment. Those segments classified as WAN segments are then anchored to the WAN map using an AtlasAnchor object created by the atlas creator; thereafter, the AtlasNode objects associated with those segments classfied as LAN segments are placed on the LAN Object list for subsequent processing.

In Step **708**, all routers in the database, along with their interfaces to various segments, are enumerated. Specifically, an AtlasNode object is created for each router and an AtlasLink object is created for each interface. If a router interfaces with one or more WAN segments, the router is anchored to the WAN map using an AtlasAnchor object.

Furthermore, if the router interfaces with one or more LAN segments, its associated AtlasNode object is placed on the LAN Object List for additional processing.

In Step **710**, the LAN Object list is examined to determine whether the list has any remaining objects. If the LAN Object list is not empty, then the atlas creator removes a first AtlasNode object from the list in Step **712** and examines this object for associated AtlasLink objects in Step **714**. If the object has no associated AtlasLink objects, the AtlasNode object is anchored to the Island map in Step **716** using an AtlasAnchor object.

If the AtlasNode object does, in fact, have associated AtlasLink objects, the atlas creator creates an AtlasPage object for a new LAN-connected area map in Step **718** and, in Step **720**, it further conducts a "breadth-first" search of linked AtlasNode objects to construct a LAN-area map containing all AtlasNode objects in the same LAN-connected portion of the network.

For this latter step, each AtlasNode object removed from the LAN Object list is examined to identify all such objects in the same LAN area. This is preferably performed by following each AtlasLink object from one AtlasNode object to an adjacent AtlasNode object, recursing through all adjacent node objects classified as LAN objects. To facilitate this search, the LAN Area list is used as a queue for traversing the connected graphs. Each adjacent AtlasNode object classified as a LAN object is removed from the LAN Object list and anchored, using an AtlasAnchor object, to the new AtlasPage object. Adjacent AtlasNode objects that are classified as WAN objects are also anchored to the new AtlasPage object using an AtlasAnchor object.

Returning to Step **710**, if the LAN Object list is empty (signifying that all of LAN-connected area maps are created) the atlas creator creates map cross-references in Step **722**; this is generally achieved by anchoring adjacent LAN-connect area maps to each other using AtlasAnchor objects. LAN-connected area maps are considered adjacent when a single WAN segment is anchored to two LAN-connected area maps. Here, the Segment list is enumerated and WAN segments are identified. For each identified WAN segment that is anchored to two separate LAN area maps, the AtlasNode objects representing the LAN area maps are anchored to each other using AtlasAnchor objects.

In Step **724**, the atlas creator creates AtlasLinkReference objects to identify links between map cross-references and other nodes on the same page. Specifically, the AtlasLinkReference objects identify those AtlasLink objects that should be depicted on a particular map. This step is preferably performed by enumerating all AtlasLink objects and identifying the AtlasNode objects at the ends of the links. All maps that each node is anchored to are identified as dependent pages. An AtlasLinkReference object is created for each occurance of a dependent page having an anchor to the same page as a node adjacent to the node on the dependent page. The AtlasLinkReference object establishes an associating link between the dependent page and the adjacent node, and their AtlasPage objects anchored thereto.

In Step **726**, the atlas layout manager enumerates all AtlasPage objects and invokes predetermined layout operations on each object. The process then ends in Step **728**. The psuedo code for the novel atlas creation process is as follows:

```
create Segment list
create LAN object list
create Campus object list
create WAN map and Island map for each segment in the
  database
  {if (segment type is SERIAL,X25,PPP,ISDN,FRAME
    RELAY,T1,etc) or (data-link protocol is not known
    and segment connects to two nodes which are
    routers)
  anchor segment to WAN map
else
  add segment to LAN object list}
    {for all routers in the database
    {clear LAN object flag for each interface
    {if interface is to a WAN segment mark as WAN router
else
  set LAN object flag}
  }if WAN router
    anchor to WAN map
  if router is a LAN object
    add router to LAN object list}
while LAN object list is not empty
  {node1=remove first node from LAN object list if node1
    does not interface to any other object anchor to Island
    page
else
  {create a new Campus (LAN connected area) map anchor
    Campus map to WAN map anchor node1 to Campus
    map
    //Do breadth 1st search to generate connected graph for
      Campus
  add node1 to Campus object list while Campus object list
    is not empty
    {node1=remove first node from Campus object list for
      each neighbor of node1
    {if neighbor is a segment and is a WAN object anchor
      object to Campus map
  else if neighbor is on the LAN object list
    {remove neighbor from LAN object list append neigh-
      bor to Campus object list anchor neighbor to Campus
      map}
    }
  }
}
}
  //Put map cross-references on adjacent maps for all seg-
    ment nodes
  {if segment node is a WAN segment
    for all neighbor nodes
      if neighbor node is not on this map
        if neighbor node's map is not anchored to this
          map
        {anchor neighbor node's map to this map
      anchor this map to neighbor node's map}
}
generate Link Reference objects for all cross-referenced
pages for all maps (Atlas Page objects) call page.Layout()
```

A user may further customize an atlas, i.e., modify it from the default arrangement generated by the atlas creator program, using a variety of filing and editing operations. These operations include:

File→New: Create a new atlas from the topology database.

File→Save: Save a modified atlas to the atlas database.

File→Save As: Save an atlas with a new name in the atlas database, without affecting other atlases.

Edit→Remove: Remove selected node or cross-reference icon(s) from a map by deleting the associated anchor

(s). Note, a node cannot be removed if it is only anchored to a single map.

Edit→Rename: Rename the selected node or, in the absence of a selected node, rename the current map.

Edit→New Map: Create a new map subordinate to the current map. If one or more node or map cross-reference icons are selected when this command is executed, the selected objects are moved to the new map and a cross-reference to the new map is anchored to the original map.

Edit→Delete Map: Delete a map, provided it is empty.

Edit→Layout Map: Layout the currently displayed map. If an object is selected, it is used as the root for layout operations; otherwise, the layout manager selects the root.

Edit→Resize Map: Shift all objects on the map so that the map appears in the upper left corner of the window with a small border between the objects and the edge of the map page.

Selection Operations: (Viewer highlights selected objects so they can be distinguished from non-selected objects.)

Select single object: Single click on an icon.

Select group of objects: Depress mouse button in white space on map, move mouse until displayed rectangle (rubber band) includes one or more icons and release button.

Add to selection: Perform one of previous two select operations while depressing the Ctrl key on the keyboard.

Drag and Drop operations: Hold mouse button down on selected object, move mouse to new location and release button.

Reorganize a map: Drag object(s) and drop on new location on the same map. Links between objects update automatically.

Move objects to another map: Drag selected object(s) from currently displayed map to a map name in the browser and drop. Removes objects from originating map and adds them to destination map.

Copy objects to another map: Above operation while holding the Ctrl key down. Doesn't affect originating map, adds objects to destination map.

Drag page reference on to map: Select a map name from the browser and drop on the current map; map cross-reference is placed at the drop point.

Rearrange map hierarchy: Drag a map name over another map name and drop; the dragged map becomes a child of the underlying map, bringing all of its subordinate maps with it. Drag to the root of the browser tree to move map to the top level of the browser tree. All children under a particular parent map (or root) are sorted alphabetically by name.

As a result of the above operations, the user may organize an atlas to any preferred arrangement of maps and objects. This new arrangement can be saved in the atlas database for future use.

Referring again to FIG. **6**, the nodes of the graphs are preferably router and segment objects that "interface" via connected graph links. Each map may refer to other maps of the atlas by way of map cross-reference objects, although a map may not cross-reference itself. The router, segment and cross-reference objects are displayed as icons **650** at various locations on a map; for example, the map name Fortune listed on the browser **615** is depicted as an icon **650***a* on the WAN map of pane **620**. The atlas manager **270** and atlas viewer **360** manage the links by automatically drawing lines connecting icons on the map.

There may be a situation where two objects are connected, yet only one is shown on the currently-displayed map while the other is present on a cross-referenced map. Here, the icon on the currently-displayed map is connected by a line to the cross-referenced icon of the appropriate map.

Features of the novel mapping tool include double-clicking of the mouse **112** when its pointer is focused on a map cross-reference icon to switch the display to the selected map. In other words, double-clicking the mouse **112** when the pointer is focused on the Fortune map icon **650***a* switches the top-level WAN map displayed on pane **620** to the corresponding subordinate-level Fortune map. In cases where there is context to maintain, the selected map is displayed without losing that context because the originating map's cross-reference is selected and scrolled to the center of the viewer window **600**. Hence, a user can move between maps quickly and easily.

The structure of the novel atlas facilitates security and natural partitioning of the internetwork topology data among various databases coupled to a plurality of distributed server nodes **200**. That is, the browser **615** and the atlas viewer window **600** provide an integrated directory of all atlas maps regardless of where they may reside in the network. A remote connection is needed only when viewing a map from a remote site. Use of the tree browser with dynamically expandable branches requires only a reference to the root map of the remote site in order to create an integrated directory of maps. When the user expands the branch of a remote site, a connection is opened to retrieve the directory of maps for that site.

The organization of the atlas further allows transparent navigation between such distributed databases, each of which may contain maps for their local domain. For example, referring to the map displayed on the pane **620**, the subordinate map represented by Fortune icon **650***a* may be stored in a database that is separate and remote from the database storing the SFO map **650***b*. Each map, together with the browser facility **615**, contain all the information needed to locate the data for the cross-referenced maps. Moreover, the hierarchy of map pages may be configured to apply security access controls so that users may be granted rights to view only certain pages of the atlas or certain entries/branches of the browser.

Other navigation features involve the use of pop-up windows and dialog boxes (not shown) on the viewer window **600**. In one embodiment, a user positions the mouse pointer over a particular node icon and, in response to clicking the mouse, a small pop-up window appears listing those maps upon which the object represented by the icon appears. In another embodiment, the pop-window appears in response to the mouse cursor being overlayed upon an icon for a predetermined period of time, e.g., 1 second. Additionally, a find function allows the user to specify a particular node using a dialog box that searches the topology database; again, a list is generated indicating all maps upon which the node appears. As previously described, the user may then select a map from the list for display on the window **600** by double-clicking on an entry in the list or by selecting the map name from the generated list.

Objects of the tool may be moved or copied to any map of the atlas by manipulating the icons **650**. For example, when moving objects to another map, the user may "drag and drop" the icon to a destination map's entry in the

browser **615**. Pressing the control (Ctrl) key of keyboard **114** while dropping the icon results in a copy operation. Cross-referencing of maps is effected by dragging a map name from the browser to an appropriate position on the map displayed on pane **620**.

Route tracing is a navigation feature that highlights specific links along a route. Here, the atlas manager **270** and viewer **360** cooperate to draw a broad yellow line along those links. This feature allows the user to view the route in the context of maps with which the user is familiar; the previously mentioned navigation features can then be employed to quickly and efficiently follow that route through the atlas.

As an extension to the route tracing feature, the novel mapping tool may be configured to highlight areas of the internetwork where specific protocols are routed. Advantages of this protocol-specific topology tracing feature include viewing of individual protocol routing in the context of the entire internetwork and maintenance of a single, integrated set of maps that are familiar to the user.

FIG. **8** is a schematic diagram of the network atlas objects (shown at **550** in FIG. **5**) used by application program **508** to represent key elements of the topology data of the atlas. In accordance with the invention, these objects include an AtlasPage object **800**, an AtlasNode object **810** and an AtlasLink object **820**. The AtlasPage object **800** represents a single map in the atlas, e.g., the map displayed on the right pane **620** of viewer window **600**. The AtlasNode object **810** corresponds to a node in a connected-graph, i.e., an object that is connected to other objects. As described further herein, the AtlasPage object **800** is preferably derived from the AtlasNode object **810** to enable, in the case of a cross-referenced page, that page to appear as a cross-referenced node on another page. In other words, the AtlasPage object is a "sub-class" of the AtlasNode object which allows the AtlasPage object to be anchored to another AtlasPage object and appear as a node on a page. The AtlasLink object **820** corresponds to a connected graph link that forms an association between two nodes.

Moreover, two additional objects are provided to define those nodes and links that are represented on various maps. An AtlasAnchor object **830** associates an AtlasNode object **810** with the AtlasPage object **800** upon which the AtlasNode is displayed. An attribute of this anchor association defines the position on the page where the node appears. An AtlasLinkRef object **840** associates at least one AtlasLink object **820** with an AtlasPage object **800**; this indicates that those constructed links appear on the constructed page.

In order to further understand the operations of the network atlas (and window) objects, it may be useful to examine their construction together with the major function routines that comprise the behavior of the objects. In examining the objects, it is also useful to examine the classes that are used to construct the objects (as previously mentioned the classes serve as templates for the construction of objects). Thus, the relation of the classes and the functions inherent in each class can be used to predict the behavior of an object once it is constructed.

The invention as described herein supports Windows® API messaging and, in particular, those messaging functions supported by various conventional generic base object classes of the MFC C++ class library. FIG. **9** shows a simplified class hierarchy diagram of one of these generic base classes, the Object class **900**, used to define atlas objects used in the invention. In fact, each of the classes used to construct the atlas objects are subclasses of Object and thus inherit any functional operators that are available from that base class. Indeed, the class Object **900** is a generic base class for all the MFC object classes shown in FIG. **9** and provides all message-handling functions that these object classes need.

For example, the class AtlasNode **902** is a subclass of the base class Object and is used to construct a node in a connected graph. The classes InetNode **904** and AtlasPage **906** are, in turn, derived from AtlasNode **902**; InetNode **904** is used to encapsulate a router or segment node in the internetwork connected graph, while AtlasPage **906** is used construct a page (map) in the atlas. This latter class is subclassed from a node because a map may appear on another map as an abstraction of a portion of the connected graph. The class InetPage **908** defines an internetwork specific page object.

The classes AtlasLink **910**, AtlasAnchor **920**, AtlasLinkRef **930** and LayoutEngine **940** are subclasses of Object **900**. Specifically, AtlasLink **910** is used to construct a connected graph link object between two node objects defined by AtlasNode **902**, and AtlasAnchor **920** is used to construct an object that denotes the presence and position of a node on a particular page (map) in the atlas. AtlasLinkRef **930** creates an object denoting the appearance of a particular link on a particular page of the atlas; such an object is generated when the atlas is created or edited for computation and communication efficiency for rendering a page in real-time. The class LayoutEngine **940** is used to construct a controller object for layout operations.

LayoutObject class **950** is another subclass of base class Object **900** and it is primarily used to create an object that abstracts at least one other object; in other words, LayoutObject creates a transient object used temporarily during a layout process. The classes LayoutHub **952** and LayoutAnchor **954** are further derived from LayoutObject **950**. LayoutHub **952** is an abstraction of a collection of layout objects that form a particular layout organization, e.g., a "hub" (parent) and "spoke" (child) organization, while LayoutAnchor **954** is used to construct a layout object that refers to a connected graph node's anchor.

FIG. **10** depicts simplified class hierarchy diagrams for various application, document and window objects of the invention. For example, the class AtlasDoc **1004**, used to construct an Atlas document object, is a subclass of the MFC document management class Document **1002**, while the class InetDoc **1006**, which is a subclass of AtlasDoc **1004**, is used to construct an Atlas document object that is specific to internetworking applications. The class MainFrame **1012** is a subclass of the MFC frame window class MDIFrameWnd **1010**; this former class is used to create a main Atlas viewer window object. Similarly, the class AtlasFrame **1022**, used to construct pane objects for the Atlas viewer windows, is a subclass of the MFC MDI child window class MDIChildWnd **1020**.

ScrollView **1030** is a MFC scrollable viewer window class from which two subclasses are derived: BrowseView **1032**, used to encapsulate a browser window pane object (e.g., see FIG. **6** at **615**), and PageView **1034**, used to create page (map) window pane objects (e.g., see FIG. **6** at **620**). Lastly, the class AtlasApp **1042** is a subclass of the MFC application class WinApp **1040** that is used to create an Atlas application object.

While there has been shown and described an illustrative embodiment for implementing the novel network mapping tool, it is to be understood that various other adaptations and modifications may be made within the spirit and scope of the invention. For example, additional system software routines may be used when implementing the invention in various

applications. These additional system routines include dynamic link libraries (DLL), which are program files containing collections of window environment functions designed to perform specific classes of operations. These functions are invoked as needed by the application program to perform the desired operations. Specifically, DLLs, which are generally well-known, may be used to interact with the application program and window manager to provide the viewer windows.

The foregoing description has been directed to specific embodiments of this invention. It will be apparent, however, that other variations and modifications may be made to the described embodiments, with the attainment of some or all of their advantages. Therefore, it is the object of the appended claims to cover all such variations and modifications as come within the true spirit and scope of the invention.

What is claimed is:

1. A network mapping tool comprising:

a management server for organizing topology data of an internetwork computing system as an atlas of maps on a network topology database; and

an atlas viewer interacting with the management server to display selected maps of the atlas on a specialized viewer window of a computer screen in a variety of views that facilitate comprehension of logical relationships between various components of the system, the viewer window comprising a browser for displaying a list of map names and a display pane for graphically displaying the contents of a selected map on the screen;

wherein said atlas viewer also interacts with said management server so as to permit user editing of said selected map in said display pane by user manipulation of the graphically-displayed contents using a pointing device, and so as to permit visual highlighting of at least certain of the graphically-displayed contents of said selected map to indicate areas where specific protocols are routed.

2. The network mapping tool of claim **1** wherein the management server comprises an atlas manager for coordinating access to the topology database for editing and viewing the atlas maps.

3. The network mapping tool of claim **2** wherein the atlas manager comprises an atlas creator for creating the atlas from the topology data and a layout manager for executing layout operations that enable display of the created atlas maps in a manner that is visually appealing to a user.

4. The network mapping tool of claim **3** wherein the browser allows the user to easily switch between maps displayed on the display pane by positioning a pointer on the corresponding map name and clicking a mouse button.

5. The network mapping tool of claim **4** wherein the contents of a selected map comprises cross-reference icons and wherein the browser further allows the user to easily switch between maps displayed on the display pane by positioning the pointer on a selected cross-reference icon and double-clicking the mouse button to display a map corresponding to the cross-reference icon without losing context of the corresponding map.

6. The network mapping tool of claim **5** wherein a selected map comprises a connected graph of linked router and segment nodes, the nodes being represented by network atlas objects.

7. The network mapping tool of claim **6** wherein the objects are further displayed as icons at various locations on the selected map and links coupling the icons are displayed as lines.

8. The network mapping tool of claim **1** wherein each map of the atlas depicts a page of the internetwork computing system, and wherein at least one top-level page illustrates a WAN structure of the system with at least one subordinate-level page illustrating structures of LAN-connected areas of the system.

9. The network mapping tool of claim **1** wherein said highlighting comprises displaying a colored line along the at least certain contents.

10. A method for constructing an atlas of maps using a network mapping tool having a topology database, the network mapping tool including a management server computer coupled to an internetwork computing system, the method comprising the steps of:

A. invoking an atlas creator component of the management server;

B. enumerating segments and routers stored in the topology database using the atlas creator;

C. classifying each enumerated segment as one of a WAN and LAN segment;

D. creating at least one WAN page and LAN-connected area page in response to Step C;

E. anchoring a router to the WAN page if the router interfaces with at least one WAN segment;

F. anchoring adjacent LAN-connected area pages to each other to generate cross-reference pages;

whereby, an atlas is constructed having at least one top-level page graphically displaying a WAN structure of the computing system and at least one subordinate-level page graphically displaying structures of LAN-connected areas of the system;

G. editing at least one of said area pages based upon user manipulation of at least one graphically-displayed component of said structures, using a pointing device; and

H. visually highlighting at least certain graphically-displayed components of said structures to indicate areas in said computing system where specific protocols are routed.

11. The method of claim **10** wherein said highlighting comprises displaying playing a colored line along the at least certain components.

* * * * *

US005822305A

United States Patent [19]

Vaishnavi et al.

[11] Patent Number: 5,822,305

[45] Date of Patent: *Oct. 13, 1998

[54] **PORT-LINK CONFIGURATION TRACKING METHOD AND APPARATUS**

[75] Inventors: **Vick Vaishnavi**, Danville; **Wallace Matthews**, Northwood; **Patrick Kenny**, Dover, all of N.H.

[73] Assignee: **Cabletron Systems, Inc.**, Rochester, N.H.

[*] Notice: The term of this patent shall not extend beyond the expiration date of Pat. No. 5,590,120.

[21] Appl. No.: **731,701**

[22] Filed: **Oct. 17, 1996**

Related U.S. Application Data

[63] Continuation of Ser. No. 550,630, Oct. 31, 1995, Pat. No. 5,590,120.

[51] **Int. Cl.**[6] **H04J 3/14**; H04L 12/26

[52] **U.S. Cl.** **370/254**; 370/252; 395/200.11

[58] **Field of Search** 370/241, 245, 370/252, 254, 255, 351, 401, 452; 340/825.07, 825.08, 825.16, 825.54; 395/200.11

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,513,411	4/1985	Fraser	370/13
4,817,080	3/1989	Soha	370/17
4,864,563	9/1989	Pavey et al.	370/94.1
5,049,873	9/1991	Robins et al.	340/825.06
5,079,765	1/1992	Nakamura	370/401
5,226,120	7/1993	Brown et al.	395/200.11
5,319,633	6/1994	Geve et al.	370/17
5,319,644	6/1994	Liang	370/452
5,384,768	1/1995	Fuji	370/351
5,412,654	5/1995	Perkins	370/94.1
5,471,399	11/1995	Tanaka et al.	370/13
5,481,674	1/1996	Mahavadi	395/200.11
5,513,171	4/1996	Ludwiczak et al.	370/13
5,568,471	10/1996	Hershey et al.	370/245

FOREIGN PATENT DOCUMENTS

0 455 442 A2	11/1991	European Pat. Off. .
WO 89/07377	8/1989	WIPO .
WO 91/15066	10/1991	WIPO .

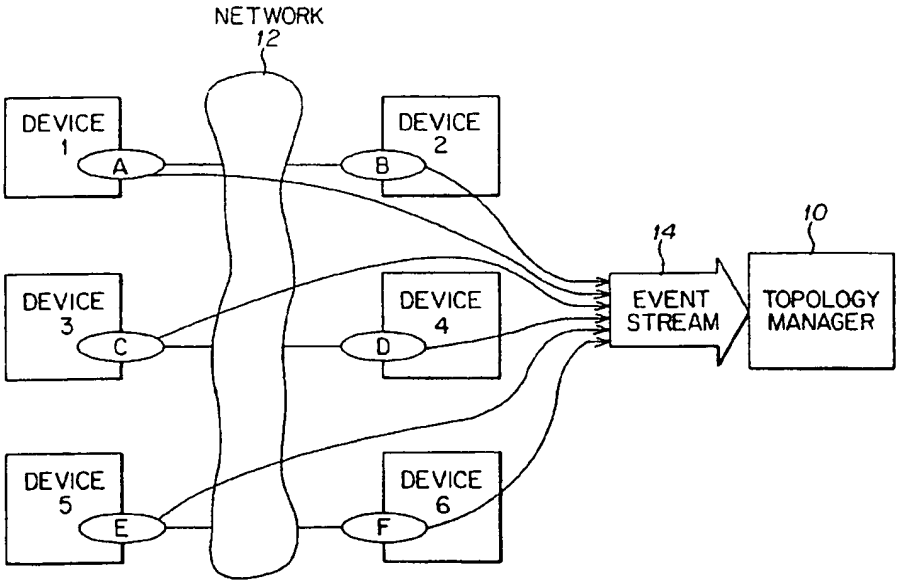
Primary Examiner—Hassan Kizou

Attorney, Agent, or Firm—Wolf, Greenfield & Sacks, P.C.

[57] **ABSTRACT**

A logical representation of a communications network topology has links which represent connections within a network, and models of ports representing elements of devices which form the connections of the network. The logical representation is created and maintained in response to reports from the network, such as new neighbor reports and lost neighbor reports. A new neighbor module creates or changes the logical representation in response to new neighbor reports, based upon whether the reporting port is recently attached and whether the new neighbor port is recently attached. A lost neighbor module changes the logical representation in response to lost neighbor reports, by creating pseudo new neighbor reports, and allowing the pseudo new neighbor reports to be processed following a certain amount of time. The operation of the new neighbor module and lost neighbor module allow reports to be processed independent of the order in which the reports are received, and also facilitates monitoring of complex network topologies, such as those including connections of more than two nodes, and those in which reports may be received in any order.

24 Claims, 11 Drawing Sheets



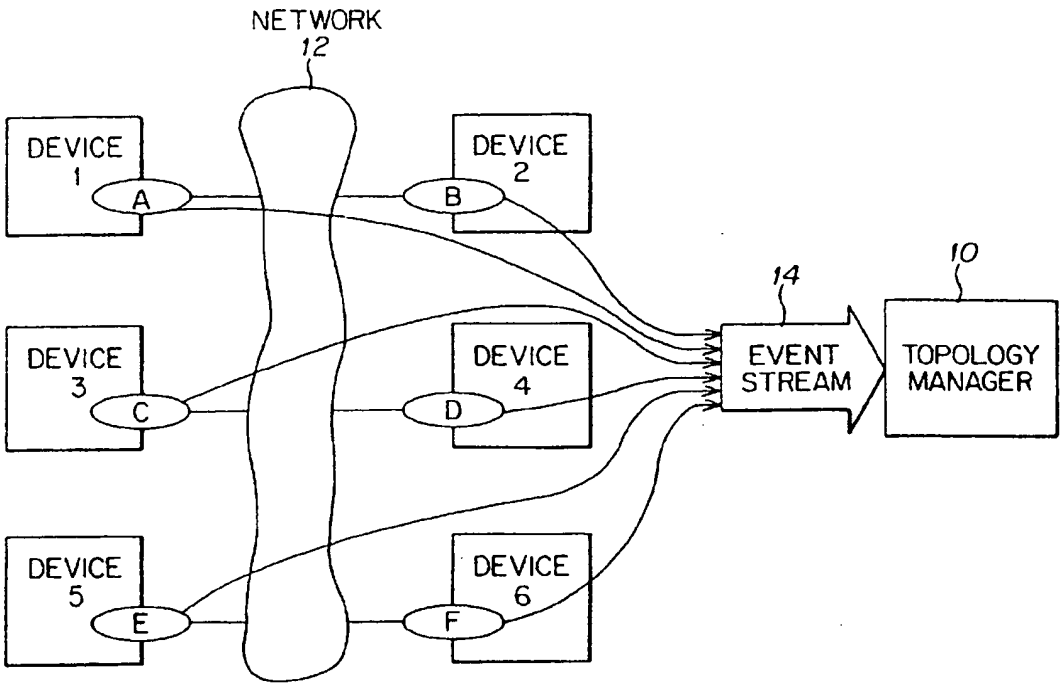

NETWORK
12
DEVICE
1
A
DEVICE
2
B
DEVICE
3
C
DEVICE
4
D
DEVICE
5
E
DEVICE
6
F
14
EVENT
STREAM
10
TOPOLOGY
MANAGER

FIG. 1

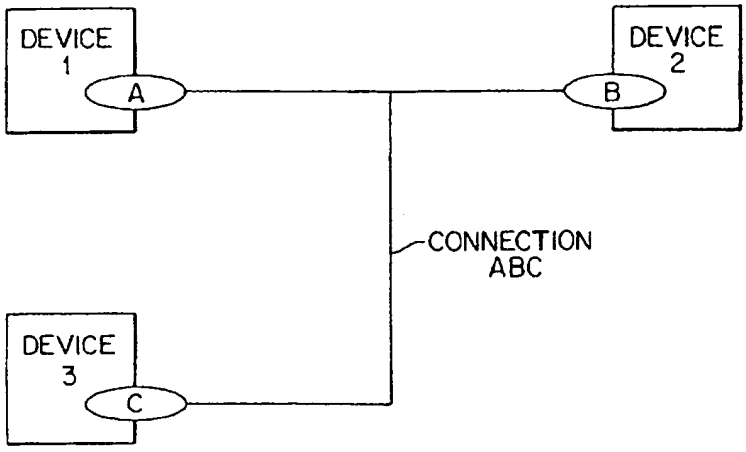

DEVICE
1
A
DEVICE
2
B
CONNECTION
ABC
DEVICE
3
C

FIG. 2

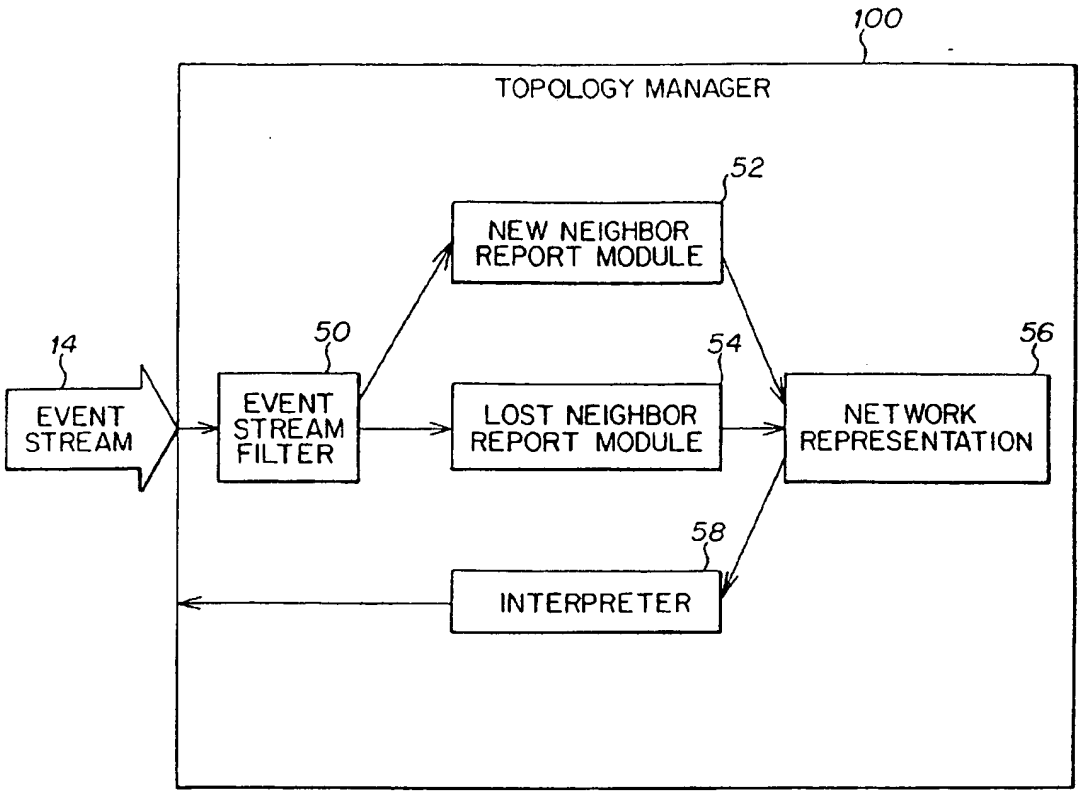

100
TOPOLOGY MANAGER
52
NEW NEIGHBOR REPORT MODULE
14
EVENT STREAM
50
EVENT STREAM FILTER
54
LOST NEIGHBOR REPORT MODULE
56
NETWORK REPRESENTATION
58
INTERPRETER

FIG. 3

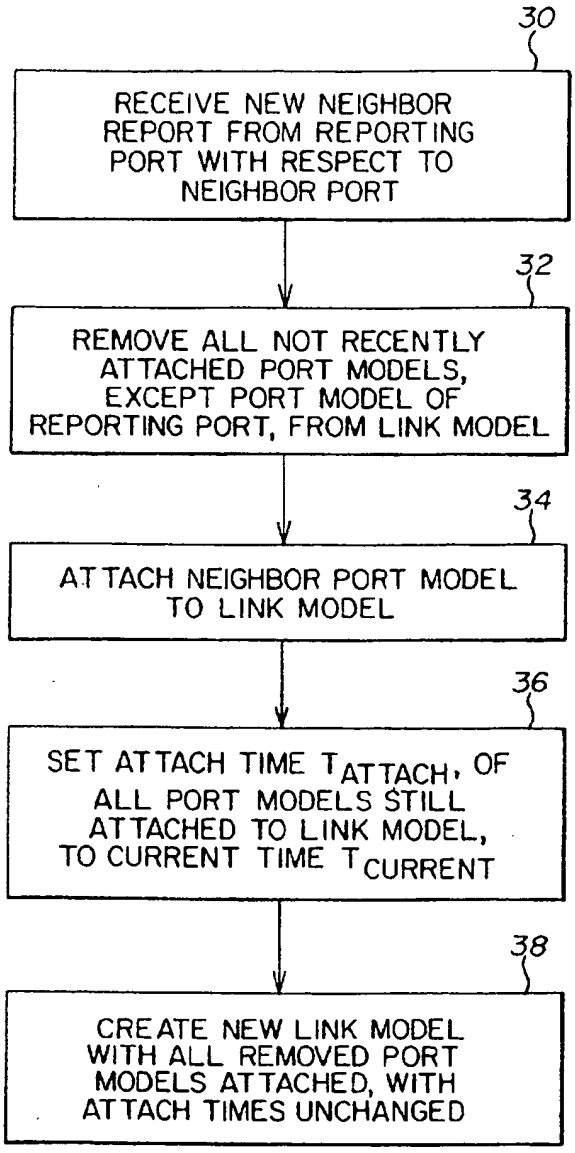

30
RECEIVE NEW NEIGHBOR REPORT FROM REPORTING PORT WITH RESPECT TO NEIGHBOR PORT
32
REMOVE ALL NOT RECENTLY ATTACHED PORT MODELS, EXCEPT PORT MODEL OF REPORTING PORT, FROM LINK MODEL
34
ATTACH NEIGHBOR PORT MODEL TO LINK MODEL
36
SET ATTACH TIME T_{ATTACH}, OF ALL PORT MODELS STILL ATTACHED TO LINK MODEL, TO CURRENT TIME $T_{CURRENT}$
38
CREATE NEW LINK MODEL WITH ALL REMOVED PORT MODELS ATTACHED, WITH ATTACH TIMES UNCHANGED

FIG. 4

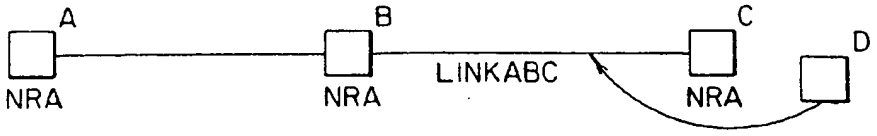


FIG. 5a

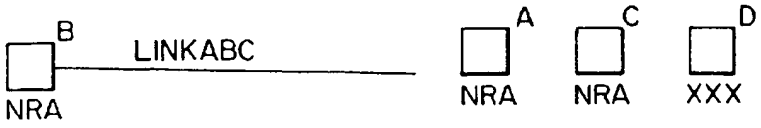


FIG. 5b

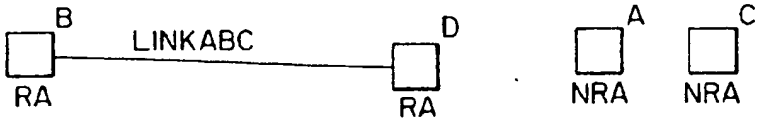


FIG. 5c

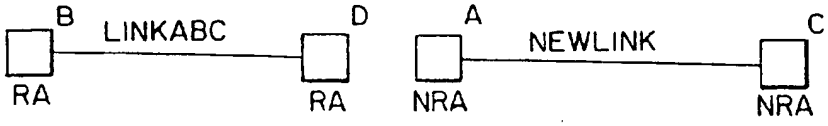


FIG. 5d

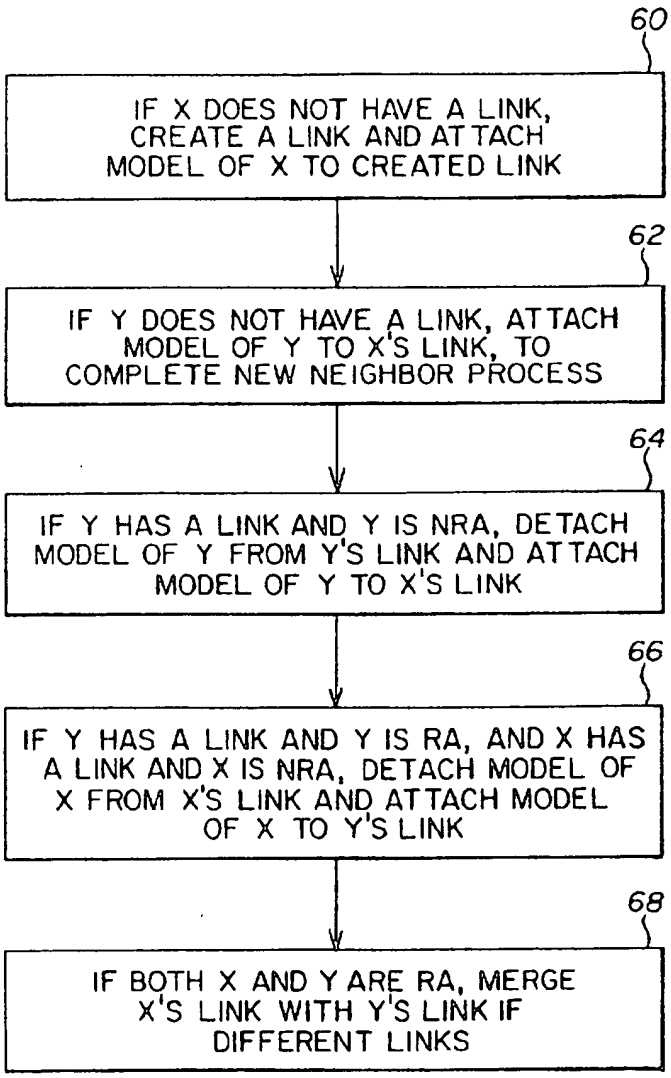

60
IF X DOES NOT HAVE A LINK, CREATE A LINK AND ATTACH MODEL OF X TO CREATED LINK
62
IF Y DOES NOT HAVE A LINK, ATTACH MODEL OF Y TO X'S LINK, TO COMPLETE NEW NEIGHBOR PROCESS
64
IF Y HAS A LINK AND Y IS NRA, DETACH MODEL OF Y FROM Y'S LINK AND ATTACH MODEL OF Y TO X'S LINK
66
IF Y HAS A LINK AND Y IS RA, AND X HAS A LINK AND X IS NRA, DETACH MODEL OF X FROM X'S LINK AND ATTACH MODEL OF X TO Y'S LINK
68
IF BOTH X AND Y ARE RA, MERGE X'S LINK WITH Y'S LINK IF DIFFERENT LINKS

FIG. 6

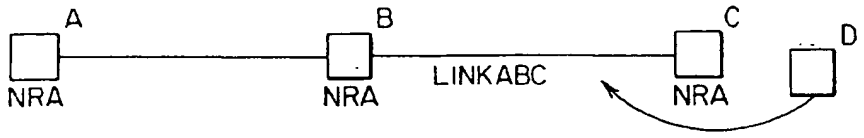


FIG. 7a

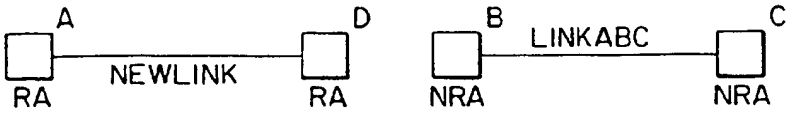


FIG. 7b

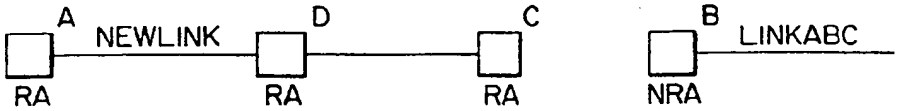


FIG. 7c

FIG. 7d

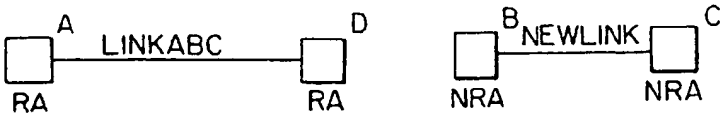


FIG. 7e

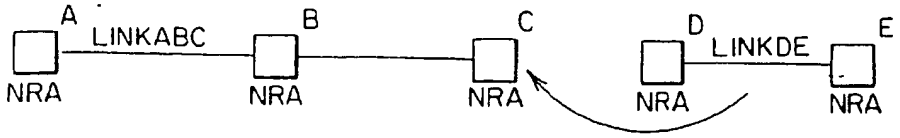


FIG. 8a

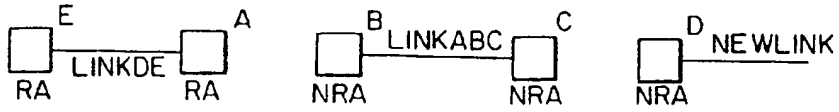


FIG. 8b

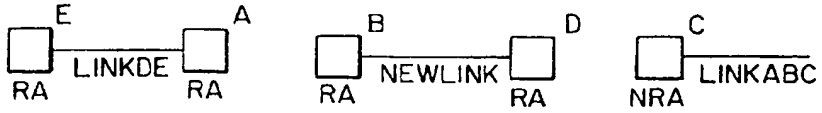


FIG. 8c

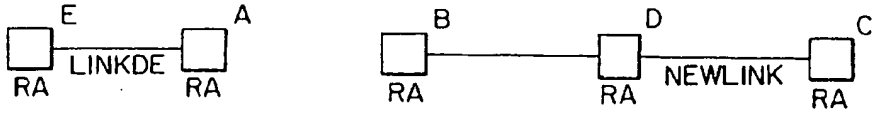


FIG. 8d

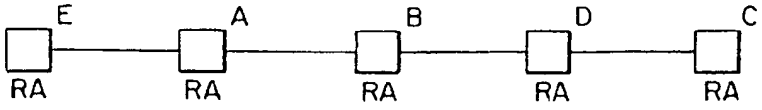


FIG. 8e

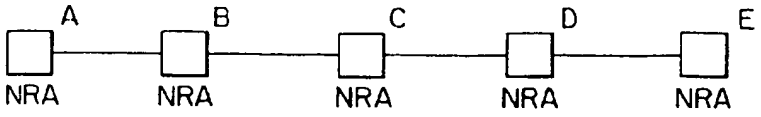


FIG. 9a

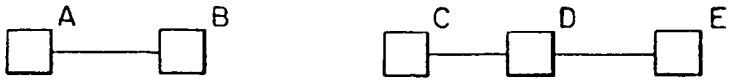


FIG. 9b

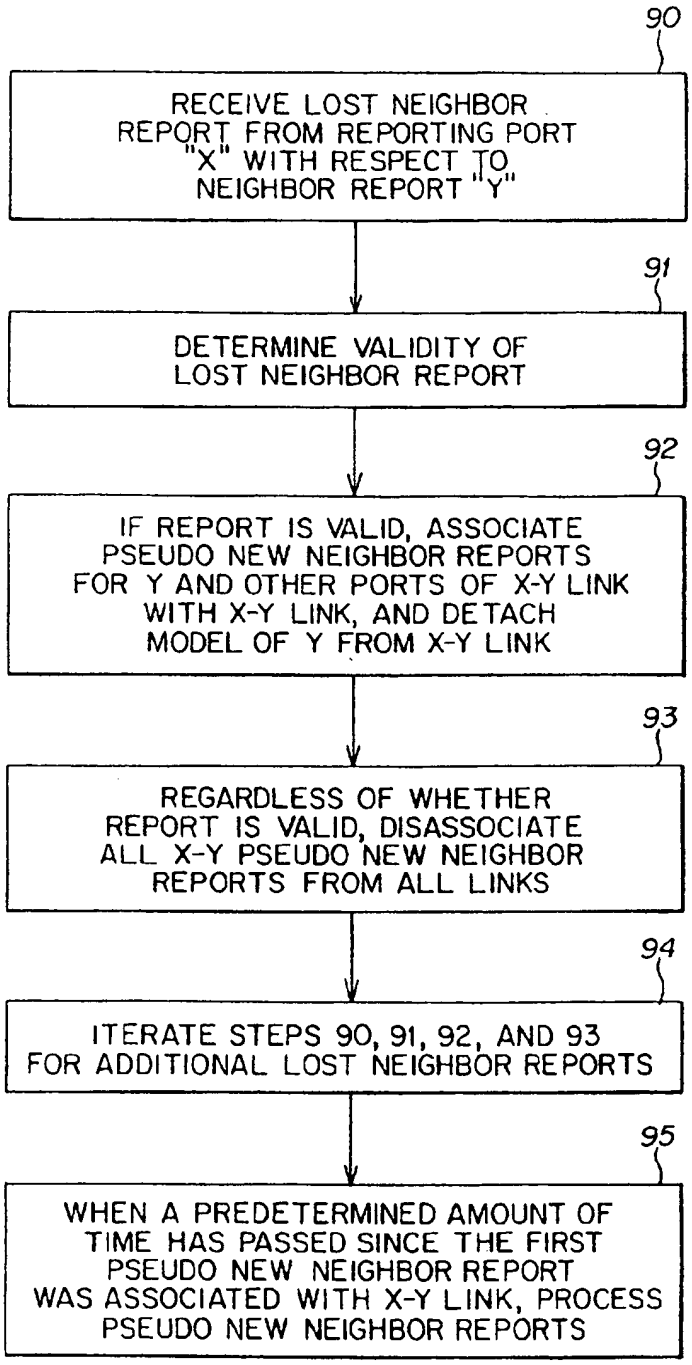

90
RECEIVE LOST NEIGHBOR REPORT FROM REPORTING PORT "X" WITH RESPECT TO NEIGHBOR REPORT "Y"
91
DETERMINE VALIDITY OF LOST NEIGHBOR REPORT
92
IF REPORT IS VALID, ASSOCIATE PSEUDO NEW NEIGHBOR REPORTS FOR Y AND OTHER PORTS OF X-Y LINK WITH X-Y LINK, AND DETACH MODEL OF Y FROM X-Y LINK
93
REGARDLESS OF WHETHER REPORT IS VALID, DISASSOCIATE ALL X-Y PSEUDO NEW NEIGHBOR REPORTS FROM ALL LINKS
94
ITERATE STEPS 90, 91, 92, AND 93 FOR ADDITIONAL LOST NEIGHBOR REPORTS
95
WHEN A PREDETERMINED AMOUNT OF TIME HAS PASSED SINCE THE FIRST PSEUDO NEW NEIGHBOR REPORT WAS ASSOCIATED WITH X-Y LINK, PROCESS PSEUDO NEW NEIGHBOR REPORTS

FIG. 10

110

REPORTER	NEIGHBOR

FIG. 11a

110

REPORTER	NEIGHBOR
A	B
A	D
A	E

FIG. 11b

110

REPORTER	NEIGHBOR
A	B
A	D
A	E
B	C
B	E

FIG. 11c

FIG. 12a

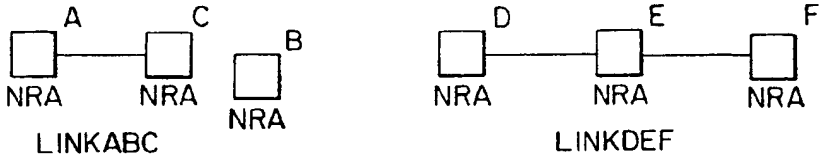


FIG. 12b

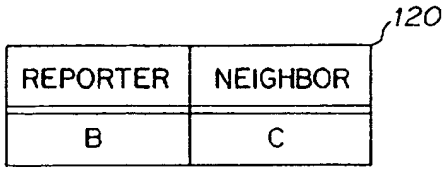


FIG. 12c

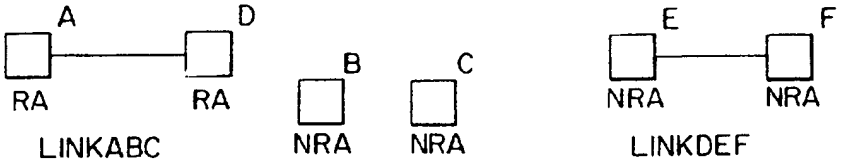


FIG. 12d

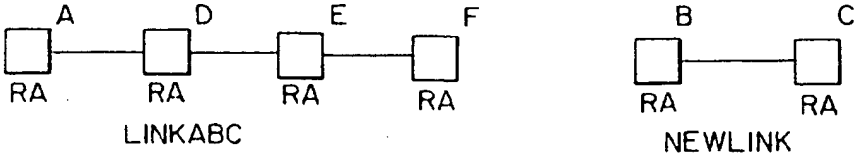


FIG. 12e

PORT-LINK CONFIGURATION TRACKING METHOD AND APPARATUS

This application is a continuation of application Ser. No. 08/550,630, filed on Oct. 31, 1995 entitled PORT-LINK CONFIGURATION TRACKING METHOD AND APPARATUS, now U.S. Pat. No. 5,590,120.

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates generally to communications networks, and more particularly to monitoring and tracking port-to-link configurations within connection-oriented communications networks.

2. Discussion of the Related Art

Communications networks provide a capability for one device, referred to as a source, to transmit data to another device, referred to as a destination. Among the conventional types of communications are connection-oriented communications and connectionless communications.

In connection-oriented communications, a logical association is established between the source and the destination, so that several separate groups of data may be sent along the same path that is defined by the logical association. This is distinguished from connectionless communications, in which the source transmits data to the destination in an unplanned fashion and without prior coordination. In connectionless communications, each frame of data is transmitted from the source to the destination in a manner independent from the manner in which other frames are transmitted from the source to the destination. Bridges and routers are commonly used in connectionless communications.

Three phases generally occur during connection-oriented communications, including connection establishment, data transfer, and connection termination. These three phases together are commonly called a session, which may be monitored and controlled by a central authority. In the connection establishment phase, the first time a source has data to be sent to a destination, a logical association, also called the connection or a path, is established between the source and the destination. The connection defines elements and connections between the elements, for example, switches between the source and the destination, and the ports of the switches through which the data will pass from the source to the destination.

A switch, and other devices similar in operation to a switch, may be referred to as a node, intermediate system, interface message processor, or gateway. A port is an interface on a switch or similar device that provides a physical communication path to other devices, for example to other ports of other switches. During the data transfer phase, data is transmitted from the source to the destination along the connection, which includes the port-to-port connections of the switches. After a certain amount of time, or at the occurrence of a certain event, the session enters the connection termination phase, in which the connection is terminated, and the elements which made up the connection are freed to support other connections.

There may be a large number of connections which represents a very large and complex amount of connection information for a central authority to monitor. Additionally, some connections may fail, due to electrical problems or physical or logical removal of any of the elements which make up the connection. For example, a switch may be removed for maintenance or to achieve a more advantageous physical arrangement for other connections. It would thus be desirable to provide a monitoring capability for tracking a large number of connections in a connection-oriented communications network.

SUMMARY OF THE INVENTION

According to several aspects of the present invention, a topology manager is provided that allows "network reports" to be processed independent of the order in which the reports are received, and also facilitates monitoring of complex network topologies, such as those including connections of more than two nodes. The topology manager includes a network representation that is a virtual model of the network topology and which is modified based upon "new neighbor reports" and "lost neighbor reports" received from the network being monitored. New neighbor reports are processed based upon whether the relevant ports of the new neighbor report are recently attached or not recently attached, so that stale or otherwise faulty reports are correctly processed. Lost neighbor reports are processed by creating pseudo new neighbor reports to be processed as actual new neighbor reports at a later time, so that the network configuration may settle prior to fully processing the reports.

In one embodiment, a method is provided for tracking a configuration of a plurality of ports of a communications network. The method comprises the steps of receiving a report which includes an indication that a reporting port has a connection to a neighbor port, determining whether the reporting port has been recently attached to any connection within the communications network, determining whether the neighbor port has been recently attached to any connection within the communications network, and determining the configuration of the plurality of ports based upon based upon whether the reporting port has been recently attached and whether the neighbor port has been recently attached. The step of determining whether either of the ports has been recently attached may include determining whether the port has been attached to a connection for a predetermined amount of time prior to the step of receiving. The method may include creating a network representation that is a virtual model of the configuration of the communications network.

The method may further include steps of receiving a second report which includes an indication that a second reporting port has lost a connection to a lost neighbor port, and the step of determining a configuration may include the steps of creating at least one pseudo new neighbor report, and processing the at least one pseudo new neighbor report to determine the configuration of the plurality of ports. The lost neighbor port and the second reporting port may be part of (connected to) a connection that includes other associated ports, and the method may include creating a pseudo new neighbor report for each of the other associated ports, the pseudo new neighbor report including the lost neighbor port as a pseudo reporting port and the each of the other associated ports as a pseudo new neighbor port. The method may further comprise the step of determining the validity of the second report by determining whether a model of the second reporting port and a model of the lost neighbor port currently share a same link within the network representation, and whether the model of the lost neighbor port has been recently attached in the network representation. The step of processing may include processing the at least one pseudo new neighbor report following a predetermined amount of time after the second report was received.

Another aspect of the invention is directed to an apparatus for tracking a configuration of a plurality of ports of a

communications network, comprising means for receiving a report that includes an indication that a reporting port has a connection to a neighbor port, means for determining whether the reporting port has been recently attached to any connection within the communications network, means for determining whether the neighbor port has been recently attached to any connection within the communications network, and means for determining the configuration of the plurality of ports based upon based upon whether the reporting port has been recently attached and whether the neighbor port has been recently attached. The means for determining whether either of the ports has been "recently" attached may include means for determining whether the port has been attached to a connection for a predetermined amount of time prior to receiving the report. The apparatus may also include means for creating a network representation that is a virtual model of the configuration of the communications network.

Another embodiment of the invention is directed to an apparatus for tracking a configuration of a plurality of ports of a communications network. The apparatus comprises a network representation, an event stream filter, a new neighbor report module, and a lost-neighbor-report module. In this embodiment, the network representation is a virtual model of the configuration of the plurality of ports. The event stream filter receives a plurality of reports from the communications network and provides a first report to the new neighbor report module and a second report to the lost neighbor report module. The first report includes an indication that a reporting port has a connection to a neighbor port, and the second report includes an indication that a second reporting report has lost a connection to a lost neighbor port. The new neighbor report module receives the first report from the event stream filter, and provides a modification of the network representation based upon the first report. Additionally, the lost neighbor report module receives the second report from the event stream filter, and provides a modification of the network representation based upon the second report.

BRIEF DESCRIPTION OF THE DRAWINGS

Other features and advantages of the present invention shall appear from the following description of certain exemplary embodiments, said description being made with reference to the appended drawings, of which:

FIG. **1** illustrates a network having connection-oriented communications, which is monitored by a topology manager;

FIG. **2** illustrates an example of a connection among three ports;

FIG. **3** shows an embodiment a topology manager in accordance with an embodiment of the invention;

FIG. **4** is a flow diagram of a process performed by a topology manager in order to attach a model of a port to a link within a network representation;

FIGS. *5a*, *5b*, *5c*, and *5d* illustrate a network representation which changes as a model of a port is attached to a link which represents a connection within the network;

FIG. **6** is a flow diagram of a process performed by a topology manager, in response to the receipt of a new neighbor report from the network;

FIGS. *7a*, *7b*, *7c*, *7d*, and *7e* illustrate a network representation which changes in response reports received from the network indicating that a port has been attached to an existing connection;

FIGS. *8a*, *8b*, *8c*, *8d*, and *8e* illustrate a network representation which changes in response to reports received from the network indicating that a port has been detached from one existing connection and attached to another existing connection;

FIGS. *9a* and *9b* illustrate a topology of a network representation in which a single connection has been split into two other connections;

FIG. **10** is a flow diagram of a process performed by a topology manager in response to the receipt of a lost neighbor report from the network;

FIGS. *11a*, *11b*, and *11c* illustrate a pseudo new neighbor table which is associated with a link within a network representation, and used in conjunction with the process of FIG. **10**.;

FIGS. *12a*, *12b*, *12d*, and *12e* illustrate a network representation which changes in response to both new neighbor reports and lost neighbor reports; and

FIG. *12c* shows the content of a pseudo new neighbor table in conjunction with the network representations of FIGS. *12a*, *12b*, *12d*, and *12e*.

DETAILED DESCRIPTION

In order to provide a monitoring capability to track connections of a connection-oriented communications network, a representation that provides a virtual (i.e., logical) portrayal of the connections within a network may be developed and maintained. Within this disclosure, the term "link" refers to a model, which may be contained within a representation of a network, of a connection between one port and at least one other port. For example, a link may refer to a connection between a first port and a second port. Such a connection may be a simple connection such as a single wire, or a more complex connection such as an ETHERNET cable and associated local area network protocol layers. Additionally, the connection may include switches or hubs along with local area networks and other devices.

FIG. **1** shows a topology manager **10** which maintains a model of a connection-oriented network, and in particular a model of the current configuration of the elements of the network **12**. As shown in FIG. **1**, the topology manger **10** receives an event stream **14** from several ports A–F. These ports A–F may be connected via a network **12**, which may contain several individual connections among the ports A–F. The event stream **14** includes reports (i.e., messages) which provide information indicative of current and past connections among the ports A–F of the network. In one embodiment of the invention, the reports include a new neighbor report and a lost neighbor report. Each of these reports includes a reporting port and is made with respect to a neighbor port. Typically, these reports are actually generated by a switch or another device which contains the reporting port, or from a source, destination, or other nodes within the network. For the purposes of this disclosure the report may be considered as being generated by the reporting port itself, so that the phrase "port A generates a report" actually means "a device on the network provides a report with port A included as the reporting port." A new neighbor report is an indication that the reporting port has been connected to the neighbor port. The expression "X→Y" represents a new neighbor report from reporting port X with respect to new neighbor Y. A lost neighbor report is an indication that a particular connection that once existed has been lost between the reporting port and the neighbor port. The expression "X←Y" represents a lost neighbor report from

reporting port X with respect to lost neighbor Y. Another type of report is a lost all neighbors report, which is indicative that a reporting port has been disconnected from all other ports. For the purposes of this disclosure, a lost all neighbors report may be considered as a plurality of individual lost neighbor reports. It is not necessary to have reports of exactly these types, as long as some information indicative of a change or a current status of the port connections is provided.

One approach to monitoring the topology of a network is to maintain a representation of the network and update the representation as reports, such as new neighbor reports and lost neighbor reports, are received. For example, if a topology manager **10** receives a new neighbor report of a reporting port with respect to a new neighbor port, then in the representation of the network, a model of the reporting port may be disconnected from any link to which the reporting port is attached, and then the model of the reporting port may be connected to a model of the new neighbor port, to form a link between the two models. Additionally, if a lost neighbor report is received from a reporting port with respect to a neighbor port, and a model of the reporting port and a model of the neighbor port both exist within the representation of the network, then the model of the neighbor port may be disconnected from the model of the reporting port.

However, this approach does not effectively support connections among more than two ports. For example, as shown in FIG. **2**, a connection ABC may be a connection between port A of device **1**, port B of device **2**, and port C of device **3**. The connection ABC may be referred to as an interswitch connection. If device C is removed from link ABC, a total of four lost neighbor reports would be received by the topology manager **10**. In particular, the event stream may include the following reports: A←C, B←C, C←A, and C←B. However, due to situations such as delays in the generation and transmission of reports, for example by device **1**, device **2**, and device **3**, the reports may be received in any order by the topology manager **10**. The delay in generating and transmitting reports may be due to communication or processing delays, or due to a situation in which the network **12** is being reconfigured in response to other reconfigurations.

In the approach described above, the topology manager **10** would contain a representation of the connection among port A, port B, and port C, as a link ABC including a model of port A, a model of port B, and a model of port C. If the report C←A is received first, then the topology manager would disconnect the model of port A from the model of link ABC, because port C is reporting port A as a lost neighbor. At this point, the representation of the network would indicate that link ABC includes port B and port C. This is an incorrect result, and exemplifies why the approach described above does not support links of more than two ports.

An embodiment of the invention is directed to a method and apparatus for tracking a configuration of a plurality of ports of a network, in which a representation of the network is created and maintained based upon an attachment time of the relevant ports. As such, an event report may be processed based upon both the current time, for example the time that the report was received by a topology manager, and the time that the relevant ports last changed status, for example the time at which most recent attachment of one of the relevant ports was attached to a link within a representation of a network.

FIG. **3** illustrates an embodiment of the invention, which is directed to a topology manager **100** that includes a new neighbor report module **52** and a lost neighbor report module **54**, both of which act upon a network representation **56**. The topology manager **100** may reside within a device that is connected to the network **12**, for example a computer workstation that operates as the central authority for controlling the connections of the network **12**. An event stream filter **50** within the topology manager **100** receives reports from an event stream **14**, and forwards appropriate reports to the new neighbor report module **52** and the lost neighbor report module **54**. An interpreter **58** accesses data within the network representation **56** in order to provide information regarding the current connection status of the network **12** from which the event stream was received.

The network representation **56** may be implemented as a software module, for example an object-oriented software module. An object representation for a connection called "X" that includes ports A, B, and C, could be represented as "link X.ports=[A,B,C]". In object-oriented terminology, this would mean that X is an instantiation of a class called links, and contains individual objects A, B, and C. Such an approach may be useful for maintaining relationships between the ports of a network, as well as maintaining attributes of the ports.

Additionally, in one embodiment, the new neighbor report module **52** and the lost neighbor report module **54** are implemented as software which controls a computer, for example a general purpose computer such as a workstation, a mainframe or a personal computer, to perform steps of the disclosed processes. Such a general purpose computer may be connected to the network in order to receive reports, and may provide commands to devices on the network in order to control the network configuration.

Alternatively, the new neighbor report module **52** and the lost neighbor report module **54** may be implemented as special purpose electronic hardware. Additionally, in either a hardware or software embodiment, the functions performed by the different elements within the topology manager **100** may be combined in different arrangements. For example, the new neighbor report module **52** may be combined with the lost neighbor report module **54** and be implemented as a single hardware or software module.

Within the network representation **56**, the model of each port on each device of the network has an associated time stamp field, referred to as attach time T_{attach}. For example, each port may be assigned an attach time (T_{attach}), indicative of the time at which the model of that port was attached to a current link within the network representation. From a current time ($T_{current}$), which may be, for example, a time at which the topology manager **100** has received a new neighbor or lost neighbor report, each port may be classified as either recently attached (RA) or not recently attached (NRA) based upon a comparison with a predetermined constant value V, as indicated in equations (1) and (2) below:

$$NRA: T_{current} - T_{attach} \geq V \quad (1)$$

$$RA: T_{current} - T_{attach} < V \quad (2)$$

In the network representation **56**, for example, each time a model of a port is attached to a model of a link, the port is assigned an attachment time T_{attach}. At any time, the topology manager **100** may compare time $T_{current}$ to time T_{attach} to determine whether the port is to be classified as RA or NRA with respect to time $T_{current}$. In one embodiment, a constant value V of 90 seconds is used, which allows for reception of the reports associated with a topology change.

Alternatively, a different value for V could be used depending upon the number of switches in a network, or on other network characteristics.

FIG. **4** shows a process performed by the topology manager **100** when a model of a port is to be attached to a model of a link within the representation of a network. Such a process may be performed in response to a step **30** of receiving a new neighbor report from a reporting port. In step **32**, all not recently attached (NRA) port models are removed from the link model to which the reporting port is currently attached, except for the port model of the reporting port. As part of step **32**, the attach time T_{attach} of each port model attached to the link model may be compared with current time $T_{current}$ to determine whether each of the port models is NRA.

In step **34**, the neighbor port model, that was reported in step **30**, is attached to the link model. In step **36**, the attach time T_{attach} of each of the port models still attached to the link model following step **32**, is set to the current time $T_{current}$. In step **38**, a new link model is created that has attached to it all of the port models removed in step **32**. Step **38** is analogous to making an assumption that the removed ports are part of another connection. The attach time T_{attach} of each of the port models removed in step **32** remains unchanged.

The following example demonstrates the attachment process of FIG. **4**. FIG. **5***a* is a graphic showing a model of link ABC including a model of port A, a model of port B, and a model of port C, each of which is NRA. The illustrated model of link ABC is exemplary of a model built in response to previously received new neighbor and lost neighbor reports. A model of port D is to be attached to the model of link ABC, which would typically be in response to the actual port D being connected to the connection among port A, port B, and port C. The connection of port D would result in a report B→D being sent to the topology manager, indicating that reporting port B is aware of new neighbor port D. The topology manager **100**, in response to this new neighbor report, may modify the model of link ABC accordingly.

In accordance with step **32**, all not recently attached port models on the link of port B (link ABC) are removed from the link model, except for the port model of the reporting port. Thus, in the example illustrated in FIG. **5***a*, since port A and port C are both NRA, both corresponding models are removed from link ABC. Since port B is the reporting port, even though port B is NRA the port B model is not removed from link ABC. The resulting configuration is shown in FIG. **5***b*.

Following step **34**, the model of port D is attached to link ABC, because port D is the neighbor port reported in the report B→D. As indicated in step **36**, the attach time T_{attach} of all port models still attached to the link model are set by the topology manager **100** to be equal to the current time $T_{current}$, so that both the port B model and the port D model are considered RA, as shown in FIG. **5***c*.

Following step **38**, a new link model ("newlink") is created as shown in FIG. **5***d*, which includes all removed port models attached (models of port A and port C), with attach times unchanged. Because the attach times of port A and port C are unchanged, both port A and port C are still considered NRA, as shown in FIG. **5***d*, which shows an arrangement at the conclusion of the attachment process. Typically, reports in addition to the B→D report discussed above would be received, resulting in an accurate model, within the topology manager **100**, of the connections within the network **12**.

An embodiment of a new neighbor process, performed by the new neighbor module **52**, is shown in the flowgraph of FIG. **6**. This process is invoked in response to the topology manager receiving a report of X→Y (port X reports new neighbor Y). As shown in step **60**, if port X does not have a link within the network representation **56**, a link is created and the model of port X is attached to the created link. In step **62**, if port Y is not currently attached to any link, then the model of port Y is attached to X's link, and the new neighbor process is completed for the X→Y report. In step **62**, the model of port Y is attached to X's link regardless of whether X's link is RA, NRA, or recently created in step **60**.

As shown in step **64**, if port Y has a link and is NRA, then the model of port Y is detached from Y's link and the model of Y is attached to X's link, regardless of whether port X is NRA or RA. As shown in step **66**, if port Y has a link and is RA and port X has a link and is NRA, then the model of port X is detached from X's link and attached to Y's link. In step **68**, if ports X and Y are both RA, then X's link and Y's link are merged if X's link is different from Y's link. Merging, in this context, refers to the process of detaching all RA ports on one link and attaching them to the other link. If X and Y are both RA and share a common link, then no action would be taken.

Table 1 is a truth table for the actions performed as a result of a report of X→Y, and shows similar results as those illustrated in FIG. **6**.

As a result of the new neighbor process as described above and shown in FIG. **6**, the order in which reports are received by a topology manager **100** will not adversely affect the network representation **56**, and connections among more than two ports may be effectively tracked.

TABLE 1

	Y HAS NO LINK	Y HAS LINK (Y'S LINK), Y IS NRA	Y HAS LINK (Y'S LINK), Y IS RA
X HAS NO LINK	CREATE NEW LINK WITH X ATTACHED, ATTACH Y TO NEW LINK	CREATE NEW LINK WITH X ATTACHED, DETACH Y FROM Y'S LINK, ATTACH Y TO X'S LINK	CREATE NEW LINK WITH X ATTACHED
X HAS LINK (X'S LINK), X IS NRA	ATTACH Y TO X'S LINK	DETACH Y FROM Y'S LINK, ATTACH Y TO X'S LINK	DETACH X FROM X'S LINK, ATTACH X TO Y'S LINK
X HAS LINK (X'S LINK), X IS RA	ATTACH Y TO X'S LINK	DETACH Y FROM Y'S LINK, ATTACH Y TO X'S LINK	MERGE X'S LINK WITH Y'S LINK IF DIFFERENT

An example of the performance of the new neighbor process is depicted with respect to a topology as shown in FIGS. **7***a*–**7***d*. In the original topology shown, shown in FIG. **7***a*, a connection represented by link ABC includes port A, port B, and port C. In this example, there has not been any recent activity with respect to ports A–C, so each of ports

A–C are labeled NRA. As described above with respect to equations (1) and (2), the topology manager **100** may determine the difference between the attach time T_{attach} of each of the ports A–C with respect to the current time $T_{current}$ to determine whether each port is RA or NRA.

As shown in FIG. **7***a*, a port D has been added to the connection represented by link ABC. Accordingly, the network representation **56** will be updated on the basis of reports received within the event stream **14**. Several reports will likely be received by the topology manager **100**. For example, report A→D will be generated by port A, because port A now senses port D as a new neighbor. Similarly, report B→D will be generated by port B, and report C→D will be generated by port C. Also, port D will generate reports D→A, D→B, and D→C, because port D now senses ports A, B, and C.

Report D→A may be the first report received by the topology manager **100** relating to the addition of port D to the connection of ports A–C. In such an instance, in accordance with step **60** of FIG. **6**, a new link "newlink" is created, and the model of port D is attached to the created link. In accordance with step **64**, because A has link ABC and A is NRA, the model of port A is detached from A's link (link ABC) and attached to X's link (link "newlink"). This results in the configuration as shown in FIG. **7***b*. Note that in accordance with the attachment process of FIG. **3**, the T_{attach} of each of port A and port D is updated to reflect the time at which the attachment was made in the network representation. Assuming that less time has passed than the time represented by the constant V, then as shown in FIG. **7***b* both port A and port D are labeled RA. However, as depicted in step **38** of FIG. **3**, the T_{attach} of each of port B and port C are unchanged, so both port B and port C remain NRA.

If report C→D is the next report received by the topology manager **100**, following report D→A, then step **66** would be followed, because port D has a link, port D is RA, port C has a link, and C is NRA. Accordingly, the model of port C is detached from C's link (link ABC), and the model of port C is attached to Y's link (link "newlink"), resulting in the configuration shown in FIG. **7***c*. Note that port C is now considered RA due to the recent attachment of C to link "newlink." At this point, if either report A→C or report D→C are received, since each of ports A, C, and D are RA, no action would be taken.

If report B→D is the next report received by the topology manager **100** following report C→D, then in accordance with step **66**, the model of port B is detached from link ABC and attached to the link "newlink." The result of this action is depicted in FIG. **7***d*, which illustrates the proper result in which all four ports A–D are on one common connection.

An advantage of the new neighbor process of FIG. **6** is that the order in which the relevant reports is received is not critical. For example, for the topology shown in FIG. **7***a*, the report A→D may be the first report received by the topology manager **100**. In such an instance, in accordance with step **62**, the model of port D is attached to link ABC, and the attach times of both port A and port D are updated to $T_{current}$ so both ports are RA. Additionally, because the attach process is followed, in particular steps **32** and **38**, the model of port B and the model of port C are both attached to a new link "newlink," and both remain NRA. Such a situation is shown in FIG. **7***e*.

At this point, if the next report received is D→A, there will be no change in the network representation. Additionally, if either of the reports D→B or B→D is the next to be received, then the model of port B will be detached from link "newlink" and attached to link ABC, with an updated T_{attach}. Similarly, if either of the reports C→D, or D→C is the next to be received, then the model of port C will be detached from link "newlink" and attached to link ABC, with an updated T_{attach}. As discussed earlier, it is an advantage of an embodiment of the invention that the network representation **56** may be maintained in a manner which does not depend upon the order in which the reports are received.

Another example of a topology change will illustrate the merging of two links, which occurs when it is determined that two connections have been formed into a single connection. For example, as shown in FIG. **8***a*, there exists a connection among ports A–C, and a connection between ports D–E. Accordingly, the network representation **56** contains a link ABC including models of ports A–C, and a link DE including models of ports D–E. In this example, there have been no recent changes to the topology, so all ports are NRA.

Any one of several reports could be received first by the topology manager **100**. Assuming, for example, that report E→A is received first, then as a result of step **64** and the attachment process of FIG. **3**, a new link "link DE" is created to which the models of ports E and A are attached. Additionally, the models of ports B and C are in a separate link ABC, with NRA status unchanged. Additionally, because the model of port D was detached from link DE, link E is in its own link "newlink." This arrangement is shown in FIG. **8***b*. If the next received report is report D→B, step **64** and the attachment process of FIG. **3** would cause the model of port B to be added to D's link "newlink," and both port B and port D would be updated to RA status, as shown in FIG. **8***c*. FIG. **8***d* illustrates the result if report C→D is the next report to be received. In particular, C has been added to the link "newlink" and updated to RA status. At this point, if any port from the connection represented by link DE becomes aware of any port from the connection represented by link "newlink," then the links will be merged. For example, if the next report received is D→A, then the final configuration will be as illustrated in FIG. **8***e*. Any additional reports received will not alter this final configuration.

An embodiment of the lost neighbor process also uses a time stamp with respect to a port, and allows representation of connections which are split into two or more connections. The conventional topology manager **10** is generally ineffective in handling the splitting of a connection. For example, FIG. **9***a* shows a connection having ports A–E, all of which are NRA. If this connection is split into a first connection including ports A–B and a second connection including ports C–E, such as shown in FIG. **9***b*, the possible reports would include A→C, A→D, A→E, B→C, B→D, B→E, C→A, C→B, D→A, D→B, E→A, and E→B. If the first report received is A→C, for example, the model of port C would be removed from the link representing the connection. If the report A→D follows, the model of port D would be removed from the link representing the connection. At this point, there is no indication that port C and port D are physically on the same connection, because no new neighbor reports will be generated with respect to ports C and D, because they have been neighbor ports all along. Therefore, ports C and D would have remained detached from each other in a network representation in the conventional art. Other examples of this drawback may be observed with other orders in which the reports may be received.

An embodiment of the lost neighbor process, typically performed by the lost neighbor module **54**, is illustrated in FIG. **10**. According to this embodiment, reports called pseudo new neighbor reports (PNNs) are generated and

associated with a link representing a connection for which a lost neighbor report has been received. Depending upon the reports received, some of these PNNs may be disassociated with the link. When a certain time period has passed, which is typically indicative that the transition from one topology to another topology has been completed, the PNNs are processed as if they were normal new neighbor reports, as described above with respect to the new neighbor process.

As shown in FIG. **10**, in step **90** a lost neighbor report is received, report X→Y, which is indicative of reporting port X reporting that neighbor port Y has been lost. In step **91**, the validity of the lost neighbor report is determined. In one embodiment, step **91** involves determining whether the models of port X and port Y currently share a same link, and whether the model of port Y is recently attached. In particular, if the models of port X and port Y do not currently share a same link, then it is possible that the report X→Y is erroneous. Therefore, the report may be deemed invalid. Additionally, if the model of port Y is classified RA, then it is possible that the report X→Y may be left over from an earlier detachment of Y, so the report may be deemed invalid.

In step **92**, if the report has been deemed valid, PNNs for port Y with respect to the other ports of the X-Y link are generated, and the model of Y is detached from the X-Y link model. In essence, the PNNs represent the assumption that other ports of the X-Y connection have not yet been disconnected, pending further information to be received from the network. In step **93**, regardless of whether the report was deemed valid in step **91**, all X-Y PNNs in all links in the network representation **56** are disassociated. In step **94**, steps **90–93** are further iterated in response to other lost neighbor reports. Additionally, during step **94**, other new neighbor reports may be processed in accordance with the new neighbor process.

Step **95** is performed when a predetermined amount of time has passed since the first PNN of the X-Y link was associated with the X-Y link. This predetermined amount of time may be based upon the amount of activity expected for a particular network. In step **95**, all of the PNNs for the X-Y link are processed as new neighbor reports. Thus, the PNNs for ports which were not detached are treated as new neighbor reports, and will result in the appropriate port models being reattached to the appropriate links.

In one embodiment, the PNNs may be contained within a table that is associated with a particular link. For example, FIG. **11***a* shows a PNN table **110** associated with the link shown in FIG. **9***a*, prior to any lost neighbor reports being received. If the connection represented by the link in FIG. **9***a* were split into the two connections represented by the links shown in FIG. **9***b*, then the reports described above with respect to FIG. **9***b* would be generated. If the first report to be received is C→A, then the process of FIG. **10** would be followed. In particular, step **92** indicates that PNNs would be generated for port A and each of the other ports of link ABCDE other than port C, and these PNNs would be associated with link ABCDE. Accordingly, PNN reports A→B, A→D, and A→E are generated and provided in the PNN table associated with link ABCDE, as shown in FIG. **11***b*.

At this point, a timer may also be initiated so that in the future it may be determined whether a predetermined amount of time has passed, as shown in step **95**. When the timer reaches a value equal to the predetermined amount of time Tpnn, then the table may be "flushed," so that all PNNs within the table are processed as if they were normal new neighbor reports.

In accordance with step **93**, any other A→C and C→A PNNs are removed from any other PNN tables which may exist within the network representation.

If the next report received is report D→B, then PNNs for port B with respect to the other reports remaining on the link ABCDE would be generated and entered into the PNN table. Since the model of port A was already removed, no PNNs are generated with respect to port A. However, because port C and port E were still attached to link ABCDE at the time that the report D→B was received, PNNs for B→C and B→E will be entered into the PNN table, as shown in FIG. **11***c*. Additionally, the model of port B is detached from link ABCDE (step **92**), and any PNN reports for B→D and D→B would be removed from other PNN tables of other links.

As shown in step **94**, the process is iterated for additional lost neighbor reports until the predetermined amount of time has passed, indicated for example by the timer reaching the value Tpnn. When a predetermined amount of time has passed, since the first pseudo new neighbor report was associated with the link X-Y, as shown in step **95**, then the pseudo new neighbor reports are processed as though they were actual new neighbor reports.

FIGS. **12***a*–**12***e* illustrate an example of the new neighbor report module **52** operating in conjunction with lost new neighbor report module **54**. As shown in FIG. **12***a*, there exists in this example a connection of ports A, B, and C, each of which is NRA. This connection is represented as link ABC. There is also another connection, represented as link DEF, that includes ports D, E, and F, each of which is NRA. Within the network, port A is disconnected from the connection with ports B and C, and connected to the connection represented by link DEF. This would not be a complicated situation if all of the new neighbor reports were guaranteed to arrive prior to the arrival of the lost neighbor reports. However, the new neighbor report module **52** and lost neighbor report module **54** operate effectively even in the presence of different arrival orders.

The following reports would be generated in such an instance: A→B, A→C, A→D, E→A, F→A, B→A, A→E, D→A, A→F, and C→A. For the purposes of this example, this arrival order will be illustrated. The first event is a lost neighbor event, A→B. In the network representation **56**, A's link and B's link are the same, and both are NRA, so in accordance with step **91** this report is considered valid. Accordingly, the model of port B is detached from link ABC, resulting in the arrangement illustrated in FIG. **12***b*. Also, appropriate entries are added to the PNN table **120** for link ABC, as shown in FIG. **12***c*.

In the order given above, the next report to arrive is report A→C. Both port A and port C share a same link, and both are NRA, so the model of port C is detached in accordance with step **92** of the lost neighbor process. No entries are added to the PNN table **120** for link ABC, because there are no other ports of the relevant link.

When report A→D arrives, the model of port D is removed from link DEF, and is attached to link ABC, which at this time has only the model of port A attached. This results in the topology as shown in FIG. **12***d*, and the PNN table **120** for link ABC remains the same.

The next two reports received in this example are report E→A and report F→A. These reports will cause the models of ports E and F to be detached and attached to link ABC. Note that if link DEF had a PNN list, then link DEF would not be removed from the network representation **56** until the associated timer had expired and the pseudo new neighbor reports were generated and processed.

After the attachments of the models of ports E and F to link ABC, the remainder of the reports do not cause any

changes, and may be considered redundant. In particular, the remaining lost neighbor reports are not valid, because the ports involved are not on the same link anymore. Additionally, the remaining new neighbor reports are redundant because the models of ports A, D, E, and F are all on the same link ABC already.

When the timer associated with link ABC reaches Tpnn, the pseudo new \neighbor report B→C will be processed, resulting in the models of ports B and C to be linked together, resulting in the topology as shown in FIG. 12*e*.

With respect to the embodiments described above, a topology manager allows network reports to be processed independent of the order in which the reports are received, and also facilitates monitoring of complex network topologies, such as those including connections of more than two nodes, and those in which reports may be received in any order.

Having thus described several embodiments of the invention, various alterations, modifications, and improvements will readily occur to those skilled in the art. Such alterations, modifications, and improvements are intended to be within the spirit and scope of the invention. For example, the network representation **56** may be maintained as a conventional database structure or with special purpose software, instead of as an object-oriented structure. Accordingly, the foregoing description is by way of example only, and not intended to be limiting. The invention is limited only as defined in the following claims and the equivalents thereto.

What is claimed is:

1. A method for tracking a configuration of a plurality of ports of a connection-oriented communications network, comprising the steps of:

(a) receiving a plurality of reports, each of the plurality of reports providing an indication of a connection status between at least two of the plurality of ports;

(b) determining that a connection exists in the connection-oriented communications network among at least three of the plurality of ports; and

(c) determining the configuration of the plurality of ports including the connection determined in step (b).

2. The method of claim **1**, wherein the step (b) includes determining a connection status in a manner that is independent of an order of which the plurality of reports are received in step (a).

3. The method of claim **1**, wherein the step (c) includes determining the configuration of the plurality of ports based upon an attachment time of at least one of the plurality of ports.

4. The method of claim **1**, wherein the step (c) includes determining the configuration of the plurality of ports based upon whether a reporting port of the plurality of reports has been recently attached, and based upon whether a neighbor report corresponding to the reporting port has been recently attached.

5. A method for tracking a configuration of a plurality of ports of a communications network, comprising the steps of:

(a) receiving a plurality of port reports, each of the plurality of port reports including an indication of a connection status between at least two of the plurality of ports; and

(b) determining the configuration of the plurality of ports based upon an attachment time for at least one of the plurality of ports as indicated in at least one of the plurality of port reports received in step (a).

6. The method of claim **5**, wherein step (b) includes classifying at least some of the plurality of ports as being either recently attached or not recently attached.

7. The method of claim **6**, wherein the step of classifying includes determining whether at least some of the plurality of ports have been attached to a connection for a predetermined amount of time.

8. The method of claim **5**, wherein step (b) includes determining the configuration of the plurality of ports independent of an order in which the plurality of reports are received in step (a).

9. An apparatus for tracking a configuration of a plurality of ports of a connection-oriented communications network, comprising:

means for receiving a plurality of reports, each of the plurality of reports providing an indication of a connection status between at least two of the plurality of ports;

means for determining that a connection exists in the connection-oriented communications network among at least three of the plurality of ports;

means for determining the configuration of the plurality of ports including the connection determined by the means for determining that a connection exists.

10. The apparatus of claim **9**, wherein the means for determining that a connection exists includes means for determining a connection status in a manner that is independent of an order of which the plurality of reports are received by the means for receiving.

11. The apparatus of claim **9**, wherein the means for determining the configuration of the plurality of ports includes means for determining the configuration of the plurality of ports based upon an attachment time of at least one of the plurality of ports.

12. The apparatus of claim **9**, wherein the means for determining the configuration of the plurality of ports includes means for determining the configuration of the plurality of ports based upon whether a reporting port of the plurality of reports has been recently attached, and based upon whether a neighbor report corresponding to the reporting port has been recently attached.

13. An apparatus for tracking a configuration of a plurality of ports of a communications network, comprising:

means for receiving a plurality of port reports, each of the plurality of port reports including an indication of a connection status between at least two of the plurality of ports; and

means for determining the configuration of the plurality of ports based upon an attachment time for at least one of the plurality of ports as indicated in at least one of the plurality of port reports received by the means for receiving.

14. The apparatus of claim **13**, wherein the means for determining includes means for classifying at least some of the plurality of ports as being either recently attached or not recently attached.

15. The apparatus of claim **14**, wherein the means for classifying includes means for determining whether at least some of the plurality of ports have been attached to a connection for a predetermined amount of time.

16. The apparatus of claim **13**, wherein the means for determining includes means for determining the configuration of the plurality of ports independent of an order in which the plurality of reports are received by the means for receiving.

17. The apparatus of claim **16**, wherein the means for determining includes a network representation, the apparatus further comprising a report module, coupled to the network representation, having an output that modifies the

network representation based upon an attachment time of at least one of the plurality of ports.

18. The apparatus of claim **16**, wherein the means for determining includes a network representation the apparatus further comprising a report module coupled to the network representation, having an input that receives a plurality of reports from the connection-oriented communications network, and an output that modifies the network representation in a manner that is independent of an order in which the plurality of reports are received.

19. The apparatus of claim **16**, wherein the means for determining includes a network representation the apparatus further comprising a report module having an input that receives a plurality of reports from the connection-oriented communications network, and output that modifies the network representation based upon whether a reporting port within the connection-oriented communications network has been recently attached and whether a neighbor port within the connection-oriented communications network has been recently attached.

20. An apparatus for tracking a configuration of a plurality of ports within a communications network, comprising:

a network representation including a data structure that is a virtual model of the configuration of the plurality of ports, the network representation having an input; and

a report module having an output that is coupled to the input of the network representation, the output of the report module modifying the network representation based upon an attachment time of at least one of the plurality of ports.

21. The apparatus of claim **20**, wherein:

the report module has an input that receives a plurality of reports from the communications network, each of the plurality of reports providing an indication of a connection status between at least two of the plurality of ports, and

the output of the report module modifies the network representation in a manner that is independent of an order in which the plurality of reports are received.

22. An apparatus for tracking a configuration of a plurality of ports of a connection-oriented communications network, the apparatus comprising:

a network representation including a data structure that is a virtual model of the configuration of the plurality of ports, the network representation including at least one representation that a single connection exists among at least three of the plurality of ports; and

a report module, coupled to the network representation, having an output that modifies the network representation based upon an attachment time of at least one of the plurality of ports.

23. An apparatus for tracking a configuration of a plurality of ports of a connection-oriented communications network, the apparatus comprising:

a network representation including a data structure that is a virtual model of the configuration of the plurality of ports, the network representation including at least one representation that a single connection exists among at least three of the plurality of ports; and

a report module coupled to the network representation, having an input that receives a plurality of reports from the connection-oriented communications network, and an output that modifies the network representation in a manner that is independent of an order in which the plurality of reports are received.

24. An apparatus for tracking a configuration of a plurality of ports of a connection-oriented communications network, the apparatus comprising:

a network representation including a data structure that is a virtual model of the configuration of the plurality of ports, the network representation including at least one representation that a single connection exists among at least three of the plurality of ports; and

a report module having an input that receives a plurality of reports from the connection-oriented communications network, and output that modifies the network representation based upon whether a reporting port within the connection-oriented communications network has been recently attached and whether a neighbor port within the connection-oriented communications network has been recently attached.

* * * * *

US005710885A

United States Patent [19]

Bondi

[11] Patent Number: 5,710,885

[45] Date of Patent: Jan. 20, 1998

[54] **NETWORK MANAGEMENT SYSTEM WITH IMPROVED NODE DISCOVERY AND MONITORING**

[75] Inventor: **Andre B. Bondi**, Red Bank, N.J.

[73] Assignee: **NCR Corporation**, Dayton, Ohio

[21] Appl. No.: **565,180**

[22] Filed: **Nov. 28, 1995**

[51] **Int. Cl.**[6] **G06F 17/00**

[52] **U.S. Cl.** **395/200.54**

[58] **Field of Search** ..... 395/200.11, 182.02, 395/200.54; 364/514 R, 514 B, 551.01; 370/241, 242, 248; 340/825.07, 825.08

[56] **References Cited**

U.S. PATENT DOCUMENTS

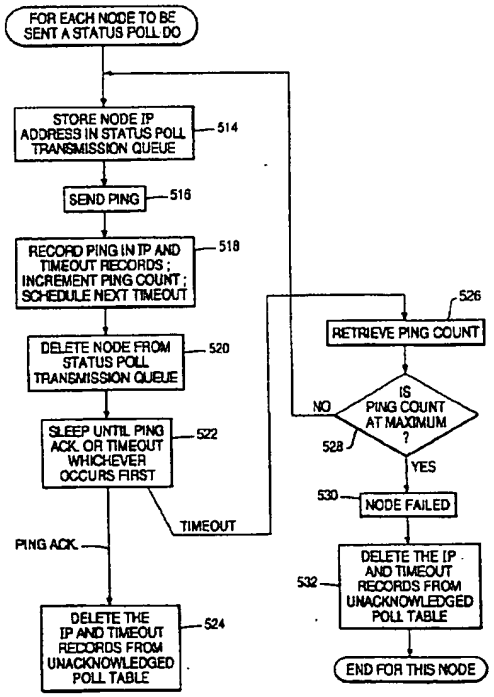
5,031,089	7/1991	Liu et al.	364/200
5,446,680	8/1995	Sekiya et al.	364/551.01
5,475,813	12/1995	Ciesiak et al.	395/182.02
5,539,883	7/1996	Allon et al.	395/200.11
5,559,955	9/1996	Dev et al.	395/182.02
5,561,769	10/1996	Kumar et al.	395/200.11
5,627,766	5/1997	Beaven	364/551.01

Primary Examiner—Ellis B. Ramirez
Attorney, Agent, or Firm—Gerard M. Wissing

[57] **ABSTRACT**

A method and system for monitoring nodes in a network having at least one network management station and a plurality of nodes is provided. Initially, polling messages are sent from the network management station to the plurality of nodes. Preferably, the polling messages are sent in a sequence whose emission is regulated at a predetermined rate. As the polling messages are sent, an unacknowledged poll table is updated with the appropriate information indicating that a set of nodes have been sent a polling message. The unacknowledged poll table is preferably indexed by an IP address of each node and is indexed by the time of the next scheduled timeout associated with each node. This method allows an arbitrary number of unacknowledged polls to be outstanding at any instant. Once the polling messages are sent and the queue is updated, the network management station then determines if a node has failed. That is, once the polling messages are sent and the queue is updated, the method verifies whether an acknowledgement is received from each of the managed nodes within a predetermined timeout period. If an acknowledgement is not received, another polling message is sent and the network management station determines if the polling message has been acknowledged after a predetermined timeout period has elapsed. This process is repeated for each node on the outstanding poll list until a predetermined number of polling messages are sent to the same target node. When the network management station sends out the last of the predetermined number of polling messages and has waited for the associated time-out period to expire, then the target node is determined to have failed.

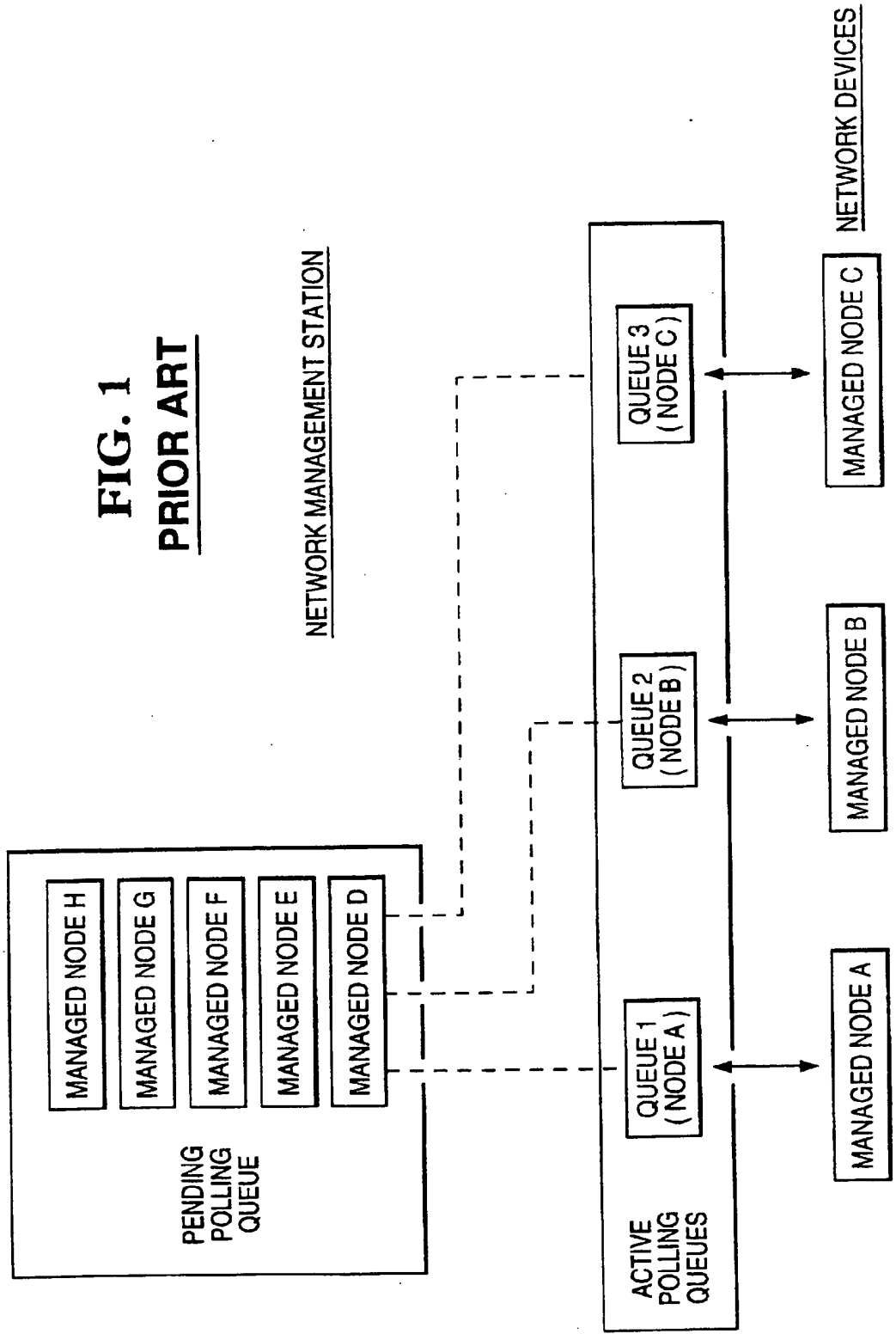
15 Claims, 7 Drawing Sheets



Polling messages predetermined time...

"PATH" NOT USED IN PATENT

FIG. 1
PRIOR ART



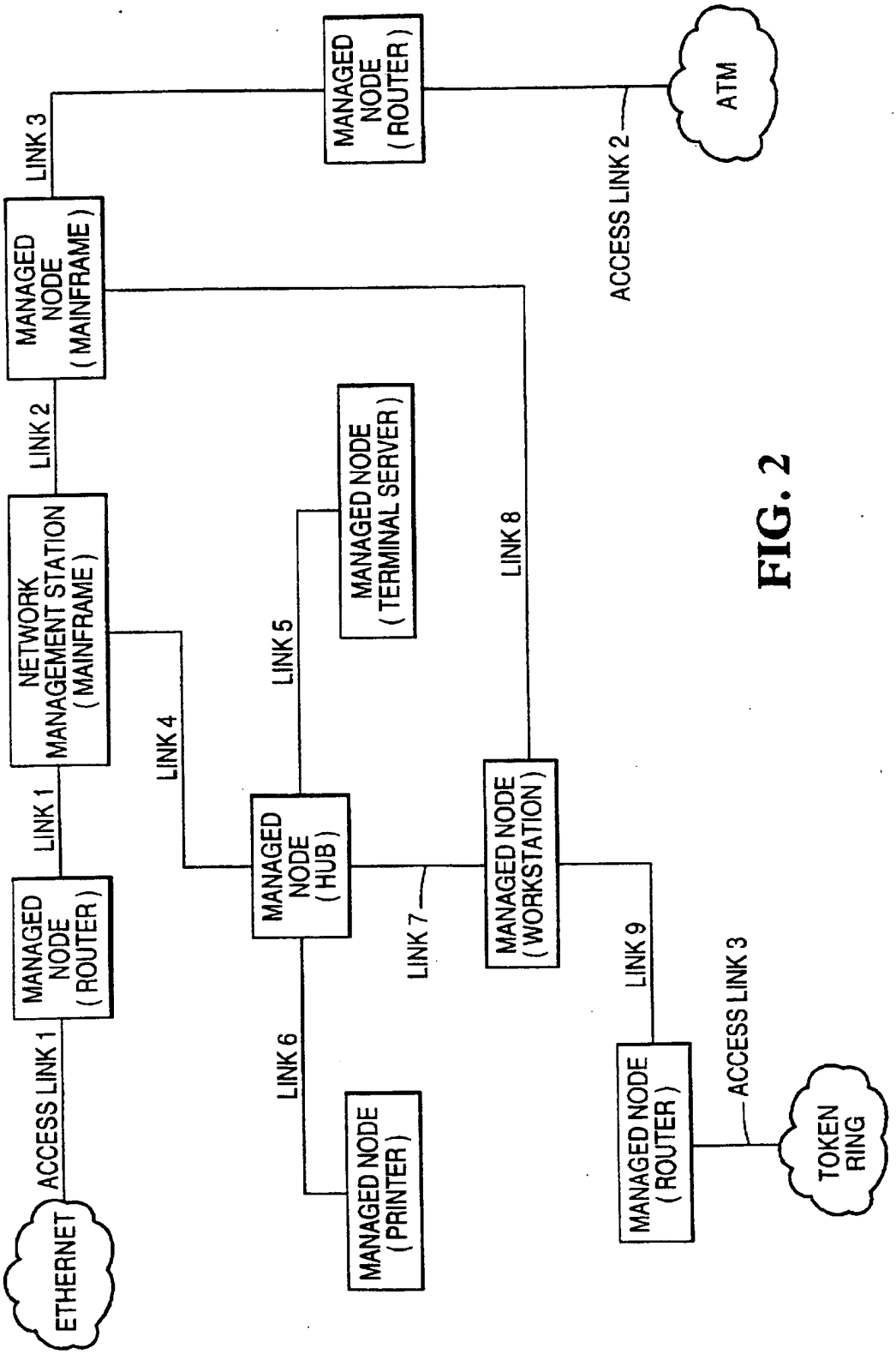

MANAGED NODE (ROUTER)
ATM
ACCESS LINK 2
LINK 3
MANAGED NODE (MAINFRAME)
LINK 2
NETWORK MANAGEMENT STATION (MAINFRAME)
LINK 1
LINK 4
LINK 5
MANAGED NODE (TERMINAL SERVER)
LINK 8
MANAGED NODE (HUB)
MANAGED NODE (WORKSTATION)
LINK 7
LINK 9
MANAGED NODE (ROUTER)
ACCESS LINK 1
ACCESS LINK 3
ETHERNET
LINK 6
MANAGED NODE (PRINTER)
TOKEN RING

FIG. 2

FIG. 3

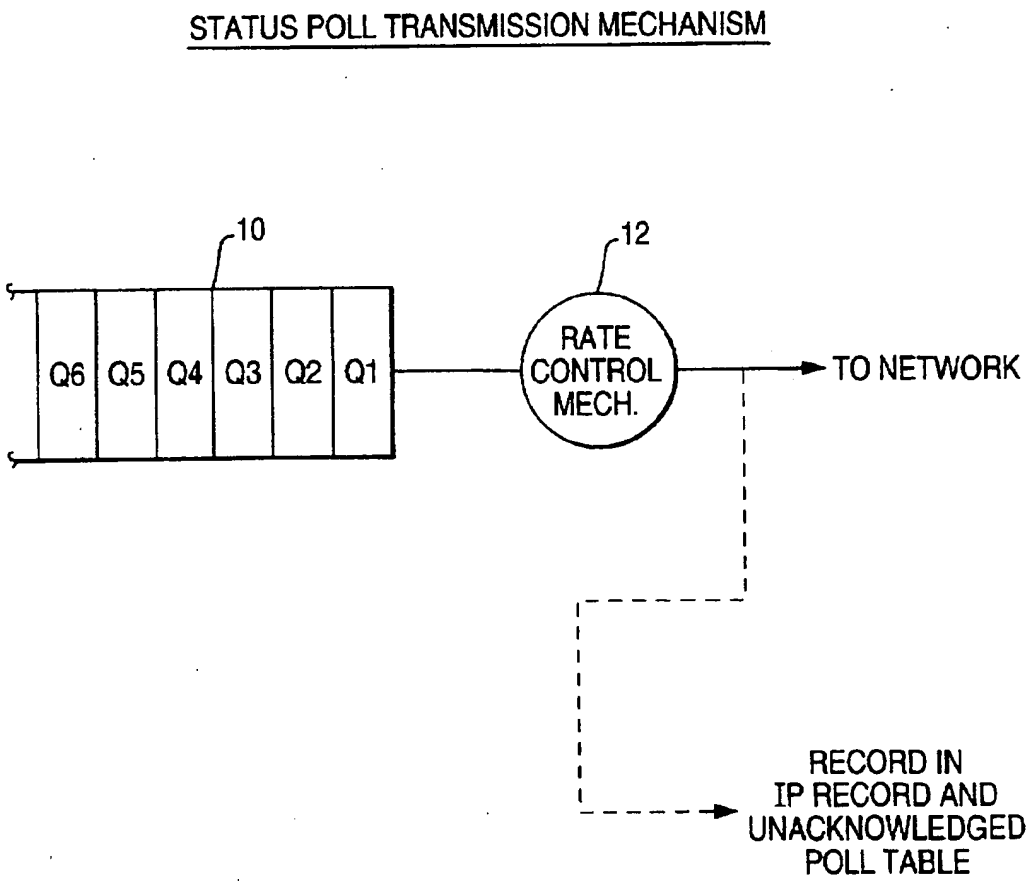
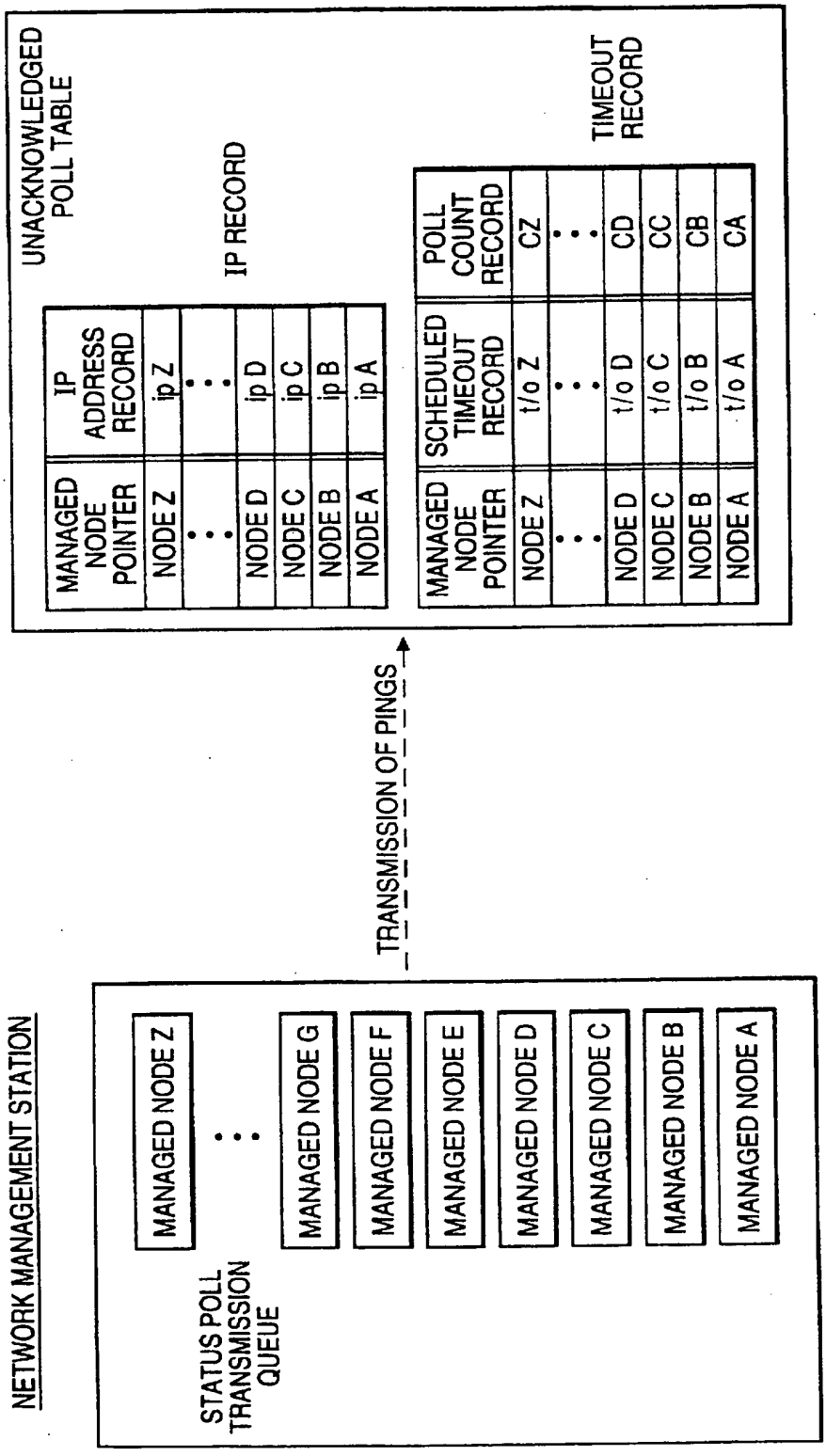

STATUS POLL TRANSMISSION MECHANISM
10
12
Q6
Q5
Q4
Q3
Q2
Q1
RATE CONTROL MECH.
TO NETWORK
RECORD IN IP RECORD AND UNACKNOWLEDGED POLL TABLE

FIG. 4


NETWORK MANAGEMENT STATION
STATUS POLL TRANSMISSION QUEUE
MANAGED NODE Z
MANAGED NODE G
MANAGED NODE F
MANAGED NODE E
MANAGED NODE D
MANAGED NODE C
MANAGED NODE B
MANAGED NODE A
TRANSMISSION OF PINGS
UNACKNOWLEDGED POLL TABLE
IP RECORD
MANAGED NODE POINTER
IP ADDRESS RECORD
NODE Z
ip Z
NODE D
ip D
NODE C
ip C
NODE B
ip B
NODE A
ip A
TIMEOUT RECORD
MANAGED NODE POINTER
SCHEDULED TIMEOUT RECORD
POLL COUNT RECORD
NODE Z
t/o Z
CZ
NODE D
t/o D
CD
NODE C
t/o C
CC
NODE B
t/o B
CB
NODE A
t/o A
CA

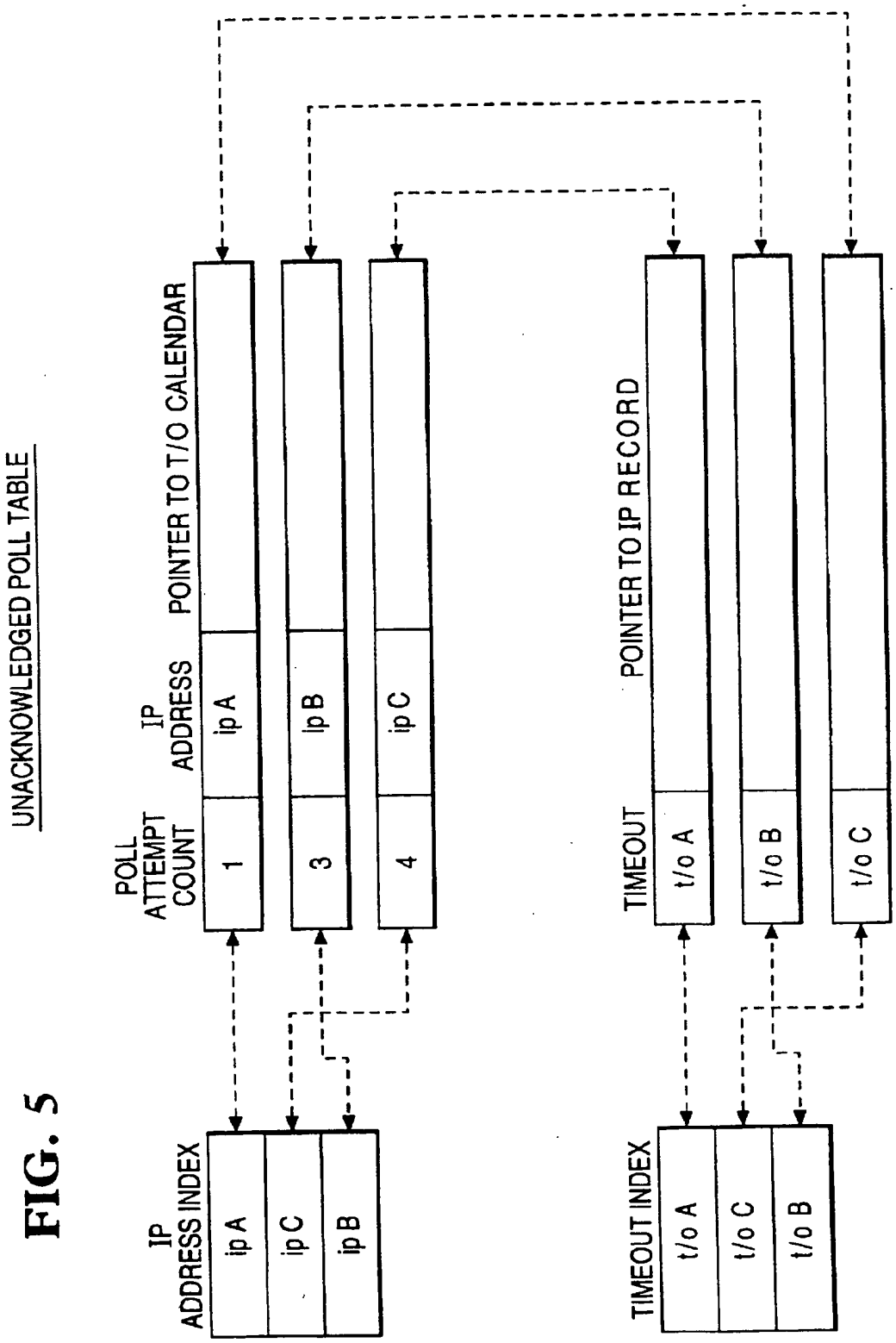

UNACKNOWLEDGED POLL TABLE
POLL ATTEMPT COUNT
IP ADDRESS
POINTER TO T/O CALENDAR
1
3
4
ip A
ip B
ip C
TIMEOUT
POINTER TO IP RECORD
t/o A
t/o B
t/o C
IP ADDRESS INDEX
ip A
ip C
ip B
TIMEOUT INDEX
t/o A
t/o C
t/o B

FIG. 5

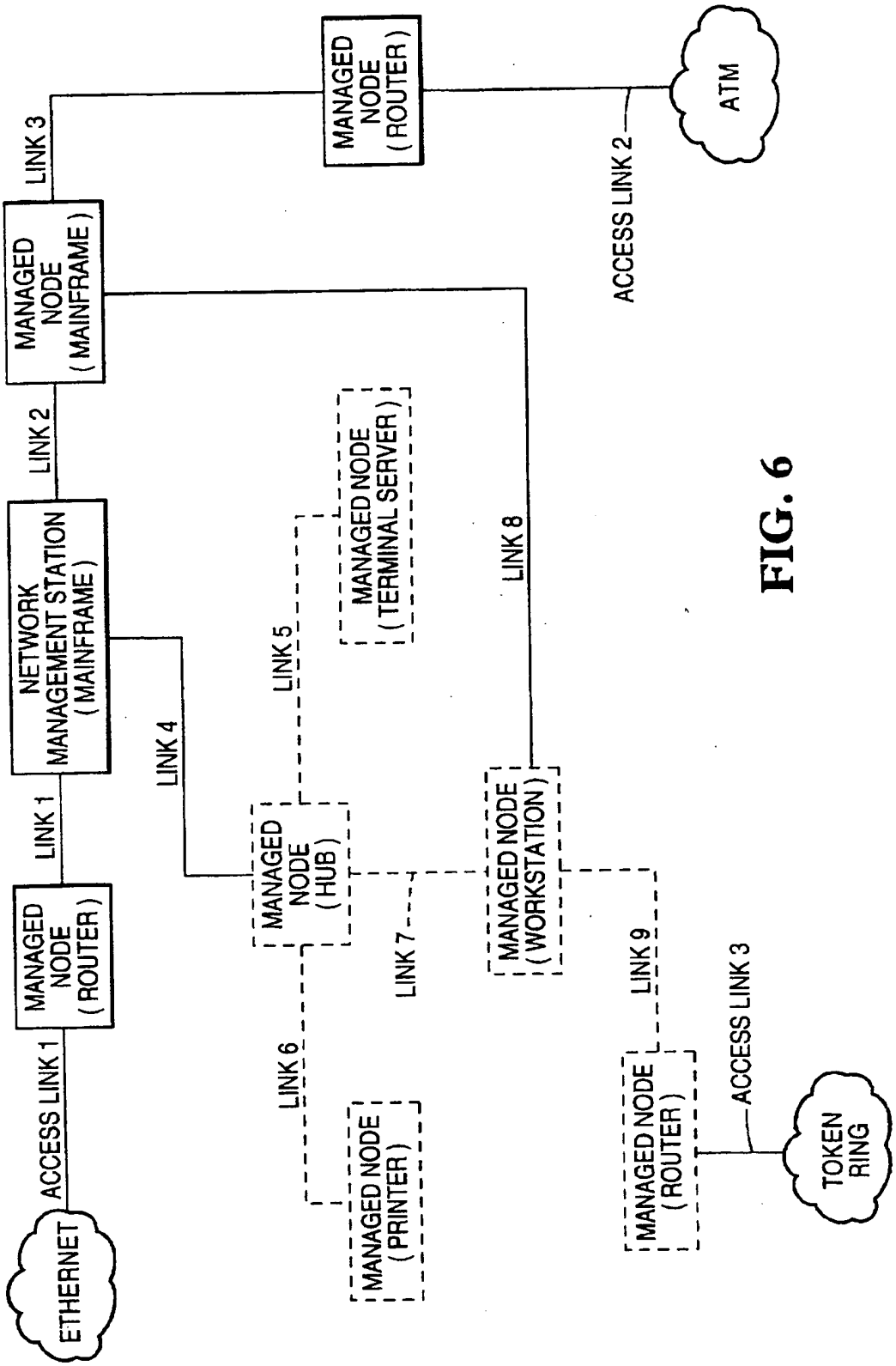
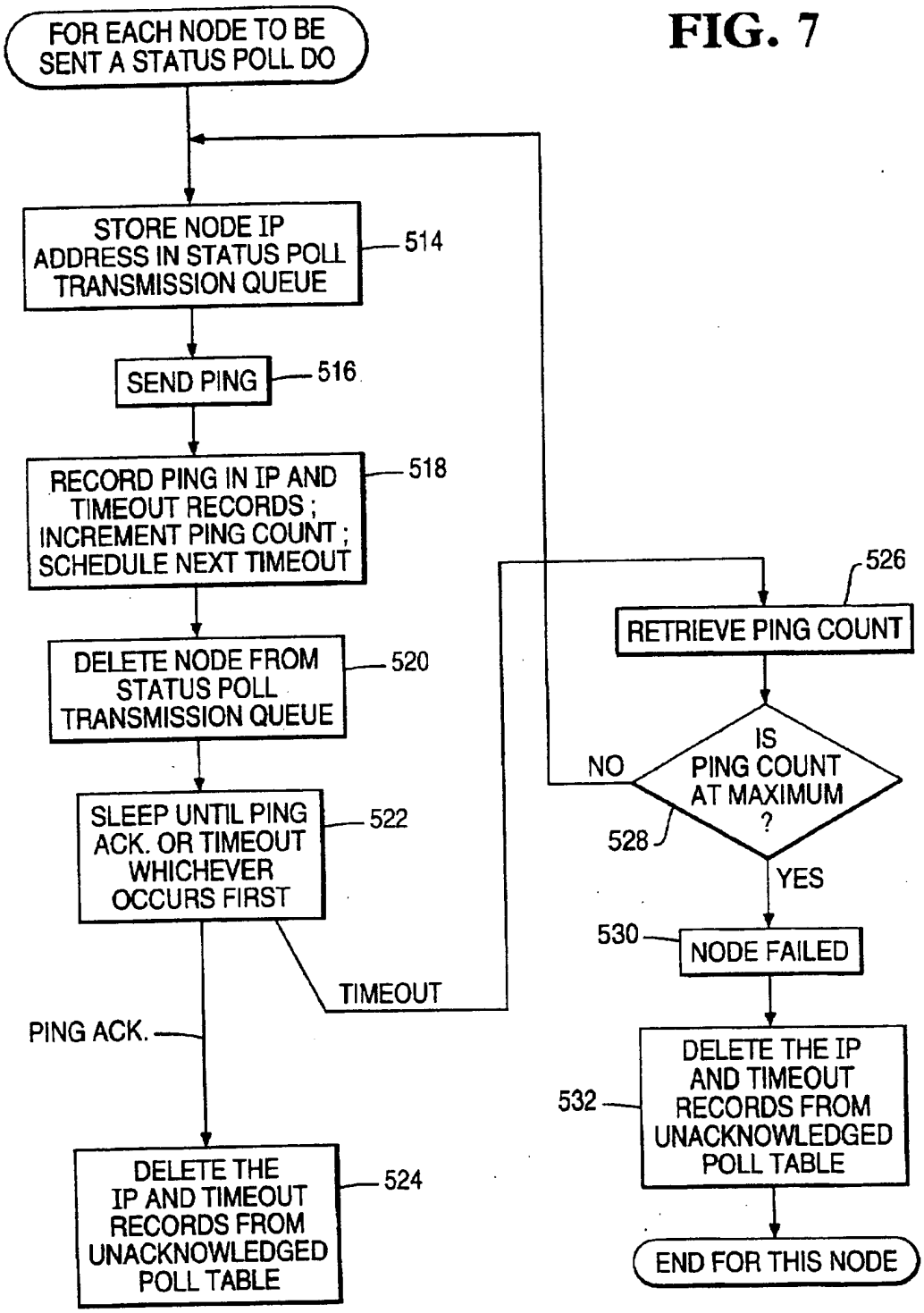

MANAGED NODE (ROUTER)
ATM
ACCESS LINK 2
LINK 3
MANAGED NODE (MAINFRAME)
LINK 2
NETWORK MANAGEMENT STATION (MAINFRAME)
LINK 1
MANAGED NODE (ROUTER)
ACCESS LINK 1
ETHERNET
LINK 4
LINK 5
MANAGED NODE (TERMINAL SERVER)
LINK 8
MANAGED NODE (HUB)
LINK 6
MANAGED NODE (PRINTER)
LINK 7
MANAGED NODE (WORKSTATION)
LINK 9
MANAGED NODE (ROUTER)
ACCESS LINK 3
TOKEN RING

FIG. 6

FIG. 7


FOR EACH NODE TO BE SENT A STATUS POLL DO
STORE NODE IP ADDRESS IN STATUS POLL TRANSMISSION QUEUE
514
SEND PING
516
RECORD PING IN IP AND TIMEOUT RECORDS ; INCREMENT PING COUNT ; SCHEDULE NEXT TIMEOUT
518
DELETE NODE FROM STATUS POLL TRANSMISSION QUEUE
520
SLEEP UNTIL PING ACK. OR TIMEOUT WHICHEVER OCCURS FIRST
522
TIMEOUT
PING ACK.
DELETE THE IP AND TIMEOUT RECORDS FROM UNACKNOWLEDGED POLL TABLE
524
526
RETRIEVE PING COUNT
IS PING COUNT AT MAXIMUM ?
528
NO
YES
530
NODE FAILED
DELETE THE IP AND TIMEOUT RECORDS FROM UNACKNOWLEDGED POLL TABLE
532
END FOR THIS NODE

NETWORK MANAGEMENT SYSTEM WITH IMPROVED NODE DISCOVERY AND MONITORING

BACKGROUND

1. Field of the Invention

The present invention relates to network management station and more particularly to a network management station which reduces the elapsed time in which a network's topology is discovered and updated.

2. Description of the Related Art

Large communication infrastructures, known as internets, are composed of wide and local area networks and consist of end-systems, intermediate systems and media devices. Communication between nodes on the networks is governed by communication protocols, such as the TCP/IP protocol. The end-systems include mainframes, workstations, printers and terminal servers. Intermediate systems typically include routers used to connect the networks together. The media devices, such as bridges, hubs and multiplexors, provide communication links between different end-systems in the network. In each network of an internet, the various end systems, intermediate systems and media devices are typically manufactured by many different vendors and to manage these multi-vendor networks requires standardized network management protocols.

Generally, to support the communication network, network management personnel want to know what nodes are connected to the network, what each node is, e.g., a computer, router, or printer, the status of each node, potential problems with the network, and if possible any corrective measures that can be taken when abnormal status, malfunction, or other notifiable events are detected.

To assist network management personnel in maintaining the operation of the internet, a network management framework was developed to define rules describing management information, a set of managed objects and a management protocol. One such protocol is the simple network management protocol (SNMP).

Network management systems need to interact with existing hardware while minimizing the host processor time needed to perform network management tasks. In network management, the host processor or network management station is known as the network manager. A network manager is typically an end-system, such as a mainframe or workstation, assigned to perform the network managing tasks. More than one end-system may used as a network manager. The network manager is responsible for monitoring the operation of a number of end-systems, intermediate systems and media devices, which are known as managed nodes. The network manager, the corresponding managed nodes and the data links therebetween are known as a subnet. Many different tasks are performed by the network manager. One such task is to initially discover the different nodes, e.g., end-systems, routers and media devices, connected to the network. After discovery, the network manager continuously determines how the network organization has changed. For example, the network manager determines what new nodes are connected to the network. Another task performed after discovery, is to determine which nodes on the network are operational. In other words, the network manager determines which nodes have failed.

Once the nodes on the network are discovered and their status ascertained, the information is stored in a database and network topology maps of the networks and/or subnets can be generated and displayed along with the status of the different nodes along the network to the network management personnel. Topology maps assist the personnel in the trouble shooting of network problems and with the routing of communications along the networks, especially if nodes have failed.

Through the discovery process, the network manager ascertains its internet protocol (IP) address, the range of IP addresses for the subnet components (i.e., the subnet mask), a routing table for a default router and address resolution protocol (ARP) cache tables from known and previously unknown nodes with SNMP agents. To ascertain the existence of network nodes, the discovery process performs configuration polls of known nodes and retrieves the ARP cache tables from the known nodes, and the routing tables. The network manager then verifies the existence of those nodes listed in these tables that it has not previously recorded in its database.

Examples of network manager systems are the OneVision™ network management station produced by AT&T and the OpenView network manager produced by Hewlett Packard. Currently, these systems discover nodes and verify the existence and status of nodes by sending to each node an internet control message protocol (ICMP) poll and waiting for a response. The ICMP poll is also known as a ping. If no response is received after a specified period of time, the node is determined to be nonoperational or to have failed. The change in status of the node is then reflected by the network management station by, for example, updating the topology map. Instances may occur when the ping is not received by the node, or the node is busy performing another task when the ping is sent. Thus, to verify that a node has actually failed, the network manager sends out a sequence of M pings, where M is an arbitrary but preferably a fixed number, such as four. Each successive ping is transmitted if a corresponding acknowledgment is not received during an associated scheduled timeout interval. Preferably, the timeout interval is increased for each successive ping. The sequence of pings terminates either if one of the pings is acknowledged, or if no acknowledgement has been received after the timeout interval associated with the Mth ping has expired. If no response is received to the Mth ping, the node is declared to be nonoperational ("down").

To illustrate, the network management station in the OpenView system sends an ICMP poll (ping) to a node and waits for a response. If no response is received from the first ping within ten seconds a second ping is sent out. If no response is received from the second ping within twenty seconds a third ping is sent out. If no response is received from the third ping within forty seconds a fourth ping is sent out. If no response is received from the fourth ping within eighty seconds the node is declared down. The total time from when the first ping is sent to the determination that the node is down can take about 2.5 minutes.

To prevent an overflow of pings from occurring during, for example, initial discovery, these current systems limit the number of unacknowledged ICMP polls to three nodes or less. To limit the number of unacknowledged polls, the ICMP polls for each managed node are stored in memory (a pending polling queue) of the network management station and subsequently transferred to an active polling queue capable of queuing only three nodes. Thus, in the example of FIG. 1, the queue for node A is in queue 1, the queue for node B is in queue 2, and the queue for node C is in queue 3. The three nodes in the active polling queue are then each polled with an ICMP poll. As a poll is acknowledged or in the event a node is declared down, the queue is cleared and

the next in line node is placed in the active polling queue. A ping is then sent to the next in line node.

Using the above queuing configuration, if for example three failed nodes are polled in rapid succession, the status of other nodes cannot be ascertained for at least the next 2.5 minutes, since no more than three nodes may have unacknowledged polls concurrently. Similarly, it may take 5 minutes to diagnose the failure of six nodes in succession. It may take 7.5 minutes to diagnose the failure of nine nodes. As a result, the discovery and/or status polling process performed by the network management station could be substantially delayed, thus increasing the elapsed time used by the network management station to perform network management tasks. Further, the topology map may be delayed in being updated, thus increasing the time to diagnose the problem with the network.

With the increase in size and use of internets, the management of such networks has become increasingly difficult. The resulting increase in the number of nodes increases the possibility of polling several failed nodes in sequence. Currently, a failure of multiple nodes would cause the discovery procedure to be effectively frozen as described above. The present invention provides an alternative technique for verifying the operational status of network nodes to reduce the elapsed time of network discovery and the elapsed time of status polling to rapidly provide network configuration updates which may be displayed on the topology map and assist network management personnel in troubleshooting failures more rapidly.

SUMMARY

A method for monitoring nodes in a network having at least one network management station and a plurality of managed nodes is provided. Initially, a plurality of node identities are arranged in a queue in an order of transmission of polling messages to the nodes. Polling messages are then sent from the network management station to the plurality of managed nodes at various intervals controlled by a rate control mechanism. Preferably, the polling messages are sent in a sequence whose emission is regulated at a predetermined rate. As the polling messages are sent, an unacknowledged poll table is updated with the appropriate information indicating that the plurality of managed nodes have been sent a polling message. The unacknowledged poll table preferably has a first portion that is indexed by a network address of each node for which an unacknowledged poll is pending and a second portion that is indexed by the time of the next scheduled timeout associated with each node for which an unacknowledged poll is pending. Once the polling messages are sent and unacknowledged polls are recorded, the network management station then determines if a node has failed. That is, once the polling messages are sent and the unacknowledged poll table is updated, the method verifies whether an acknowledgement is received from each of the nodes within a predetermined timeout period. If an acknowledgement is not received, another polling message is sent and the network management station determines if the polling message has been acknowledged within a predetermined timeout period. This process is repeated until a predetermined number of polling messages are sent to the same target node. When the network management station sends out the last of the predetermined number of polling messages and has waited for the associated timeout period to expire, then the target node is determined to have failed and is removed from the unacknowledged poll table and the polling queue.

The present invention also provides a managed network that includes at least one network management station, and a plurality of nodes connected to the network management station. The network management station includes a polling queue for arranging an order of transmission of polling messages from the at least one network management station to the plurality of nodes, and a poll table having a first portion indexed by a network address of each of the plurality of nodes and a second portion indexed by the time of the next scheduled timeout associated with a polling message count. In this configuration, the network management station can send out polling messages to a plurality of nodes concurrently and determine if each of the plurality of nodes has failed after a predetermined number of polling messages have been sent to each node.

Preferably, the network management station determines if a node has failed by determining if a polling message count has reached a predetermined number, and if a timeout period for a particular polling message count has expired. If these events are determined to have occurred then the node is determined to have failed and is removed from the unacknowledged poll table.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the invention are described hereinbelow with reference to the drawings wherein:

FIG. 1 is a block diagram of a known polling queue for determining the status of nodes;

FIG. 2 is a block diagram of an exemplary network topology;

FIG. 3 is a block diagram of the status poll transmission mechanism according to the present invention;

FIG. 4 is a block diagram of the status poll transmission queue according to the present invention;

FIG. 5 is a block diagram of an unacknowledged poll table according to the present invention;

FIG. 6 is a block diagram of the exemplary network topology of FIG. 2, illustrating a failed managed node and other nodes and links affected by the failed node; and

FIG. 7 is a flow diagram for the operation of the network management station during discovery and status verification.

DETAILED DESCRIPTION

The present invention provides a network management method and system which improves the discovery process and the status monitoring process of current network management systems. It should be noted that the following description is based on a communication network using the TCP/IP communication protocol and a network managing framework using the SNMP protocol. However, the invention is applicable to network management environments based on other network configurations using other types of communication and management protocols as well.

As noted above, during node discovery or during node status verification, the network manager sends ICMP polls (pings) to each node identified in, for example, the ARP cache and any known router tables. The node manager then waits for a response from the target node. The response may include information, such as the node IP address, status information regarding the node, the type of node (e.g., computer, router or hub), and the number of interfaces in the node. When a response is received, the node information is stored in an IP topology database. Typically, the decision to manage a node is made by network management personnel or the network management station.

Preferably, the IP topology database consists of a series of tables, each containing specific information regarding the

network. For example, a network table contains information associated with each network in the topology. This information may include the type of network, IP address, subnet mass, and the times associated with the creation and modification of the network entry in the table. A segment table contains information associated with each segment (or subnet) in the network topology. This information may include the name of the subnet, number of interfaces connected to the subnet, and the times associated with the creation and modification of the subnet entry in the table. A node table contains information associated with each node in the network topology. The node information may include, for example, the IP network manager, a SNMP system description, the number of interfaces in the node, and times associated with the creation and modification of the node entry in the table. The information stored in the IP topology database is primarily obtained from the discovery process, but may also be entered from network management personnel.

From the IP topology database, an IP topology map can be created. The IP map is a map of the network topology which places the discovered nodes in appropriate subnets, networks, and/or internets depending upon the level of the topology being mapped. In the system of the present invention, the IP map is preferably updated as the status of a node changes. The IP map displays the different nodes using icons or symbols that represent the node from, for example, an SNMP MIB file.

As discussed above, some current network management systems limit the number of unacknowledged pings to three nodes so as to prevent flooding the network with pings. FIG. 1 is a block diagram of the queuing sequence for sending pings to different nodes. As seen in the FIG., queues 1, 2 and 3 store the ping count for nodes A, B and C respectively. The queues are not cleared until the ping is acknowledged or when the time for each ping expires, i.e., a timeout occurs for the Mth ping, and the node is declared to have failed. Thus, a ping cannot be sent to node D until one of the queues is cleared.

The network manager according to the present invention provides a status poll transmission queue which speeds the processing of acknowledgements by storing unacknowledged pings in an ordered data table of arbitrary size indexed by the IP address of each target node. The size of the data table may be fixed or it may vary. To speed the management of timeouts, unacknowledged pings are also stored in an ordered data table indexed by the time by which a timeout is scheduled to occur for a particular ping. Each record in each data table contains a pointer to a corresponding record in the other table to facilitate rapid removal of the managed node from the queue in the event a timeout occurs for the Mth ping, or upon receipt of an acknowledgement of the ping, whichever occurs first.

FIGS. 3–5 illustrate a status poll transmission mechanism and queue for nodes A–Z, where A–Z represent the identity of each node assigned to the node manager. The status poll transmission queue 10 identifies the nodes which are scheduled to be polled. The status poll transmission queue 10 stores the node identity of the nodes which are awaiting transmission of a poll, and is preferably a FIFO (first in first out) queue or a FCFS (first come first serve) queue. However, other types of queues may be utilized, e.g., a LCFS (last come first serve) queue. A queue might also be ordered by some attribute of the objects waiting in it, such as priority class or node type. A rate control mechanism 12 controls the rate at which the pings are sent on the network to the nodes. As the pings are sent, records of the transmission of the pings are stored in an unacknowledged poll table, seen in FIGS. 4 and 5. As noted, the unacknowledged poll table consists of two data records (an IP record and a timeout record) that are configured to allow an arbitrary number nodes to be polled concurrently without receiving an acknowledgement. This configuration allows many status polls to be outstanding (unacknowledged) at one time. The rate control mechanism 12 prevents the network from being flooded with pings. Combining the utilization of the unacknowledged poll table configuration with the rate control mechanism 12 allows the network to be discovered rapidly even when status polls are unacknowledged for long periods of time. As seen in FIG. 5, the IP record is indexed by the IP address of the target nodes, and the timeout record is indexed by the scheduled timeout for the particular ping being transmitted. The timeout record also includes a ping count record. The scheduled timeout is the time period between successive pings targeted at a particular node. The ping count record represents an arbitrary number of pings that have been sent to the target node before the node is determined to have failed. The maximum ping count may be set by network management personnel or, more usually, by the designer of a network management system. Various factors, such as the acknowledgement return time and the probability of packet loss, are considered when determining the ping count. The acknowledgement return time is the time it takes for the acknowledgement to be received by the network management station.

The scheduled timeout may be set to a fixed, predetermined period of time between each ping. Preferably, the scheduled timeout between pings varies depending upon the ping count. For example, in a configuration where the ping count is four, the scheduled timeout between a first ping and a second ping may be set to about ten seconds, the timeout between the second ping and a third ping may be set to about twenty seconds, the timeout between the third ping and a fourth ping may be set to about forty seconds, and the time between the forth ping and the declaration of a failed node may be set to about eighty seconds.

Once a prescribed sequence of timeouts has been recorded by the network management station, the node is declared to have failed and the change in status of the network is stored in the IP topology database and reflected in the IP map. FIG. 6 illustrates an exemplary network topology map wherein the hub and its associated managed nodes were determined to have failed to acknowledge the pings.

During the discovery process the IP addresses of new nodes arrive in bulk on retrieved list (ARP cache) causing status polling requests (pings) of previously unknown nodes to be generated in bursts. To prevent the consequent pings messages from flooding the network, the system of the present invention regulates the transmission of the pings. That is, the system of the present invention schedules the pings for transmission in rapid succession at a controlled rate which may be user specified. The controlled rate of ping transmission may be dependent upon various factors including, for example, the current payload on the network, the current spare capacity on the network, and the buffer size in the portion of the kernel of the network management station's operating system that supports network activity. Preferably, the rate is no faster than that at which the kernel (i.e., the portion of the operating system of the network management station that supports process management and some other system functions) can handle acknowledgments. Alternatively, the rate may be automatically adjusted as the ability of the kernel to handle acknowledgments changes. For example, if the spare capacity of the network increases,

or if the payload on the network decreases, the rate at which pings may be sent also may be increased. Alternatively, if the spare capacity of the network decreases, or if the payload on the network increases, the rate at which pings may be sent may also be decreased.

As noted, to prevent a flood of pings on the network the pings are scheduled for transmission in rapid succession at the controlled rate using, for example, the rate control mechanism. One method for monitoring the throughput of pings is similar to the "leaky bucket" monitoring algorithm used to provide a sustained throughput for the transmission of ATM cells in an ATM network. A description of the leaky bucket algorithm can be found in "Bandwidth Management: A Congestion Control Strategy for Broadband Pocket Networks-Characterizing the Throughput-burstiness Filter", by A. E. Eckberg, D. T. Luan and D. M. Lucantoni, Computer Networks and ISDN Systems 20 (1990) pp. 415–423, which is incorporated herein by reference. Generally, in the "leaky bucket" algorithm, a set number of pings are transmitted within a specified time frame, and pings in excess of this number can be queued. As noted, the controlled rate can be set by network management personnel or can be automatically be adjusted by the network management station.

FIG. 7 is a flow diagram of the operation of the network management station during discovery and status verification. Initially, in discovery the network management station receives ARP caches and router tables from various nodes on the network via a configuration poll. The ARP caches and routing tables provide the network management station with, for example, the IP address of nodes along the network. The information obtained from the ARP cache and the routing tables is then stored in an IP topology database. As noted, the determination to manage the node is made by the network management station or network management personnel.

To verify the status of nodes, the IP addresses of the known nodes are stored in, for example, a status poll transmission queue (seen in FIG. 3) which identifies the nodes that are to be polled (step 514). When the network management station is performing status verification tasks, pings are sent to the newly discovered nodes and nodes identified in the status poll transmission queue at the designated IP addresses (step 516). As discussed above, the pings are sent in a controlled sequence at a predetermined rate.

As the pings are sent, the IP address associated with each polled node is stored in IP record of an unacknowledged poll table. Simultaneously, a ping count record in a timeout record of the unacknowledged poll table is incremented by one and the time of the next timeout is scheduled to be the current time plus the time out associated with the new ping count (step 518). Thereafter, the node is deleted from the status poll transmission queue (step 520). Once the ping is sent and the node is deleted from the queue, the system goes into a sleep mode with respect to the particular node until the ping is acknowledged or a corresponding timeout occurs, whichever occurs first (step 522). For each node in the newly retrieved ARP cache that is not known to the network management database, a status poll (ping) is sent in accordance with step 514 above. If the ping has been acknowledged, the network management station preferably deletes the IP record and timeout records in the unacknowledged poll table (step 524).

If the scheduled timeout for a ping occurs first, the network management station retrieves the ping count from the ping count record (step 526) and determines if the ping count matches the predetermined number of counts, i.e., the station determines if the ping count is at the maximum number (step 528). If the ping count does not match the predetermined count number, the IP address for the node is stored in the status poll transmission queue (step 514) and a new ping is sent to the same target node and the network management station repeats the steps, as shown in FIG. 7.

If at step 528 the ping count does match the predetermined count number, then the node is determined to have failed (step 530). Thereafter, the IP topology database is updated with the change in status of the node. The record for that node is then removed from the status poll transmission queue and unacknowledged poll table (step 532).

This process can be performed concurrently for many nodes thus reducing the delay until each managed node is polled and increasing the currency of the IP topology map.

It will be understood that various modifications can be made to the embodiments of the present invention herein without departing from the spirit and scope thereof. For example, various types of network managers and managed nodes may be used in the network topology and various system configurations may be employed. Moreover, the subject matter of the present invention may be applied to network management systems using communications protocols other than IP and network management protocols other than SNMP. Therefore, the above description should not be construed as limiting the invention, but merely as preferred embodiments thereof. Those skilled in the art will envision other modifications within the scope and spirit of the invention as defined by the claims appended hereto.

What is claimed:

1. A method for monitoring nodes in a network having at least one network management station and a plurality of nodes, comprising:

queuing the plurality of nodes in a queue in a network management station so as to arrange an order of transmission of polling messages to the plurality of nodes;

sending a polling message from the network management station to the plurality of managed nodes in sequence at a predetermined rate;

recording in the network management station transmission of the polling messages in a table having a first portion indexed by a network address of each node and a second portion indexed by a timeout associated with a polling message count for each node having an outstanding status poll; and

determining if each of the plurality of nodes has failed after a predetermined number of polling messages have been sent to each node.

2. The method according to claim 1, wherein said step of determining if the nodes have failed comprises:

determining if a polling message count has reached the predetermined number; and

determining if an elapsed timeout period for a particular polling message count has expired, such that when the polling message for a node is unacknowledged and the polling message count reaches the predetermined number and the timeout has expired the node is determined to have failed.

3. The method according to claim 2, wherein said polling message count is four and the elapsed timeout period is about 2.5 minutes.

4. The method according to claim 1, wherein the polling message is an ICMP polling message.

5. The method according to claim 1, wherein the network address is an IP address.

6. The method according to claim 1, wherein said predetermined number of polling messages is four.

7. A method for monitoring nodes in a network having at least one network management station and a plurality of managed nodes, comprising:

queuing the plurality of nodes on the network in a queue in a network management station so as to arrange an order of transmission of polling messages to the plurality of nodes;

sending a polling message from the network management station to the plurality of nodes on the network in sequence at a predetermined rate;

recording in the network management station transmission of the polling messages in a table having a first portion indexed by a network address of each node and a second portion indexed by the time of the next scheduled timeout associated with a polling message count; and

determining if another polling message should be sent to each of the nodes, said step of determining including determining if a polling message count has reached a predetermined number, and determining if the timeout period for a particular polling message count has expired, such that when the polling message for a node is unacknowledged and the polling message count reaches the predetermined number and the timeout has expired the node is determined to have failed.

8. The method according to claim 7, wherein the polling message is an ICMP polling message.

9. The method according to claim 7, wherein said polling message count is four and the elapsed timeout period is about 2.5 minutes.

10. The method according to claim 7, wherein the network address is an IP address.

11. The method according to claim 7, wherein said predetermined number of polling messages is four.

12. A managed network comprising:

at least one network management station; and

a plurality of nodes connected to the network management station for data communications therebetween;

wherein the network management station includes a queue for arranging an order of transmission of polling messages from the at least one network management station to the plurality of nodes and a poll table having a first portion indexed by a network address of each of the plurality of nodes and a second portion indexed by a timeout associated with a polling message count; and

wherein said network management station determines if each of the plurality of nodes has failed after a predetermined number of polling messages have been sent to each node within an elapsed timeout period.

13. The managed network according to claim 12, wherein the network management station determines if a node has failed by determining if the polling message count has reached a predetermined number, and determining if a timeout period for a particular polling message count has expired, such that when the polling message for a node is unacknowledged and the polling message count reaches the predetermined number and the timeout has expired the node is determined to have failed.

14. The network according to claim 13, wherein said predetermined number is four.

15. The network according to claim 13, wherein said elapsed time is about 2.5 minutes.

* * * * *

US005606669A

United States Patent [19]

Bertin et al.

[11] **Patent Number: 5,606,669**

[45] **Date of Patent: Feb. 25, 1997**

[54] **SYSTEM FOR MANAGING TOPOLOGY OF A NETWORK IN SPANNING TREE DATA STRUCTURE BY MAINTAINING LINK TABLE AND PARENT TABLE IN EACH NETWORK NODE**

[75] Inventors: **Olivier Bertin**, Nice; **Jean-Paul Chobert**, Carros; **Alain Pruvost**, Valauris, all of France

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **447,999**

[22] Filed: **May 23, 1995**

[30] **Foreign Application Priority Data**

May 25, 1994 [EP] European Pat. Off. 94480048

[51] **Int. Cl.**[6] **G06F 13/00**

[52] **U.S. Cl.** **395/200.15**; 395/200.02

[58] **Field of Search** 395/200.01, 200.15, 395/200.02; 370/94.3, 94.2, 94.1

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,905,233	2/1990	Cain et al.	370/94.1
5,079,767	1/1992	Perlman	370/94.3
5,138,615	8/1992	Lamport et al.	370/94.3
5,245,609	9/1993	Ofek et al.	370/94.3

FOREIGN PATENT DOCUMENTS

0404423 12/1990 European Pat. Off. .

OTHER PUBLICATIONS

IBM Technical Disclosure Bulletin vol. 35, No. 1B, Jun. 1992, New York US pp. 16–18 'Distributed Tree Maintenance'.

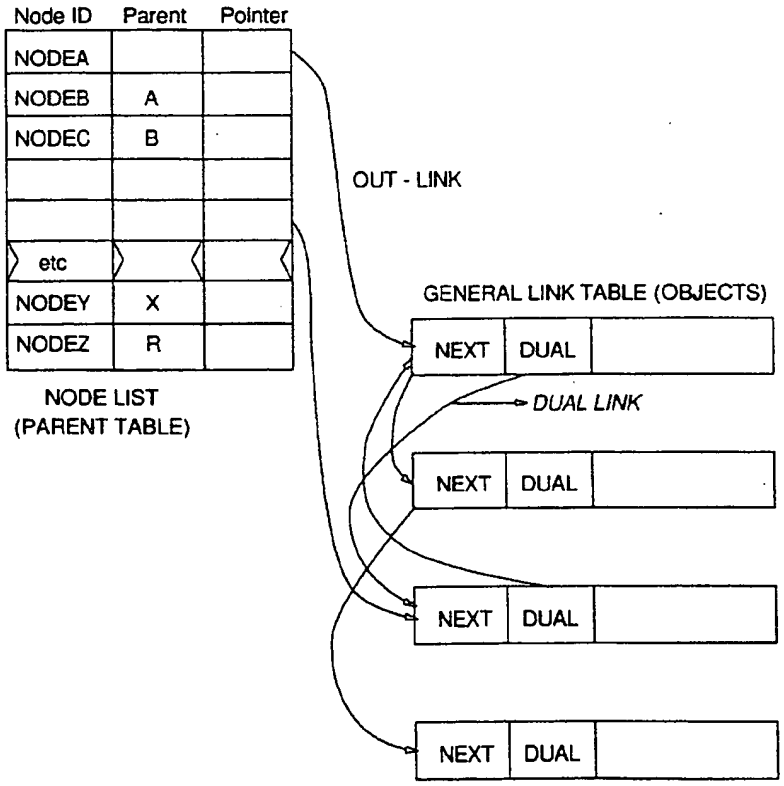
IBM Technical Disclosure Bulletin vol. 35, No. 1A, Jun. 1992, New York US pp. 93–98 'Distributed Tree Maintenance'.

Primary Examiner—Thomas C. Lee
Assistant Examiner—Le Hien Luu
Attorney, Agent, or Firm—Gerald R. Woods

[57] **ABSTRACT**

A topology manager within a data communication network including a number of nodes interconnected by bi-directional links, wherein each said node is provided with means for dynamically setting and storing within the node a full topology database including full parent-node-relationship references. The system is is capable of fast path determination and fast spanning tree recovery based on the topology database contents.

8 Claims, 9 Drawing Sheets



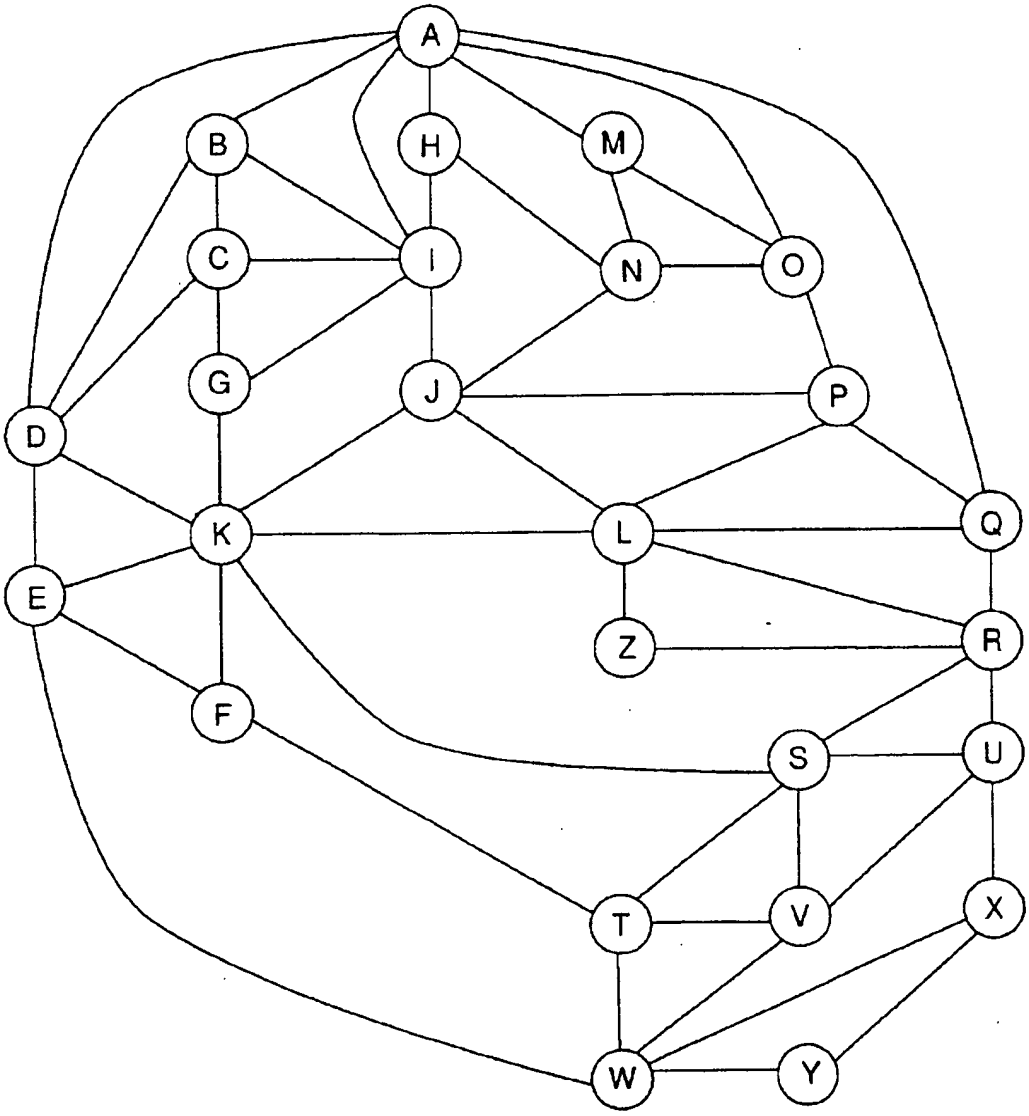


FIG. 1

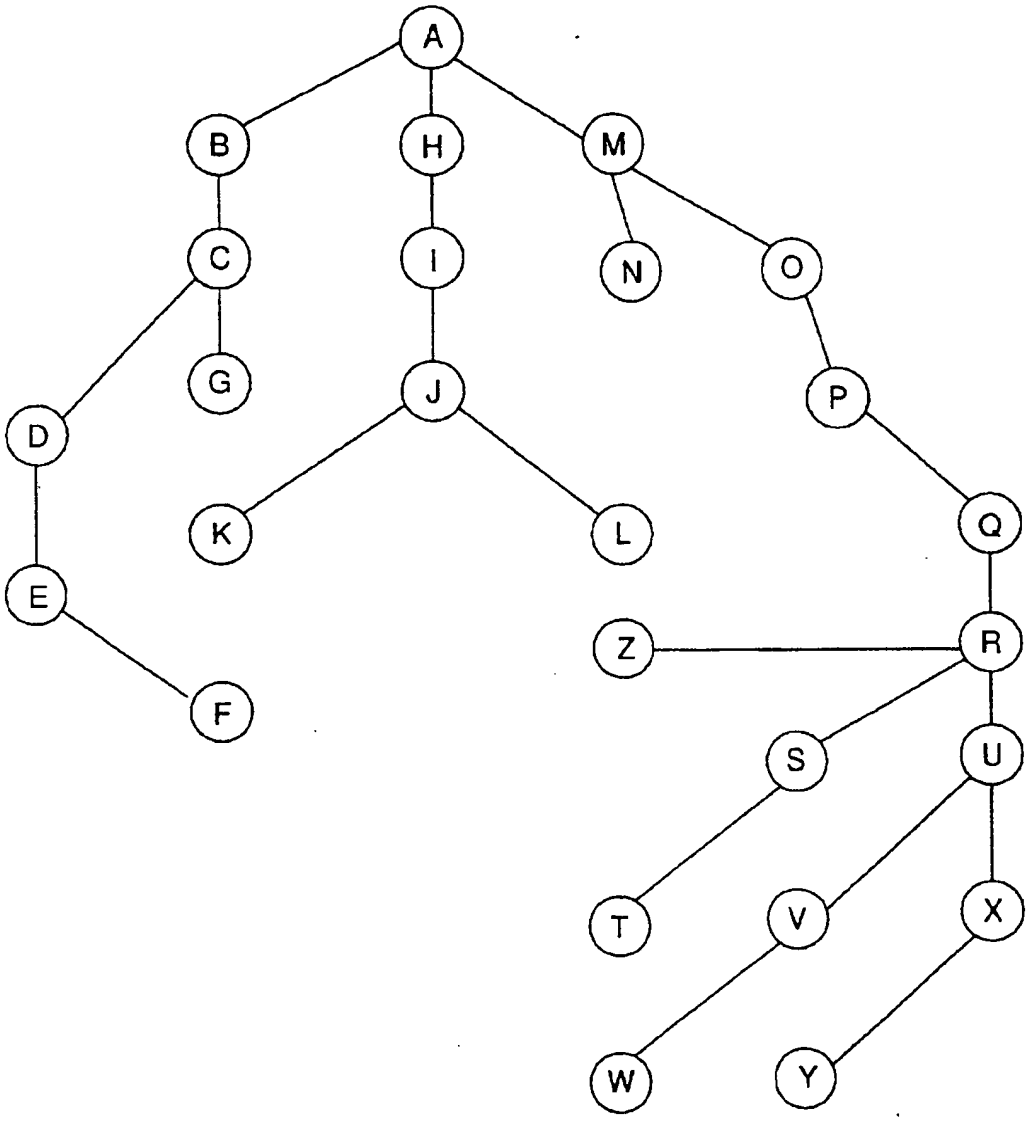


FIG. 2

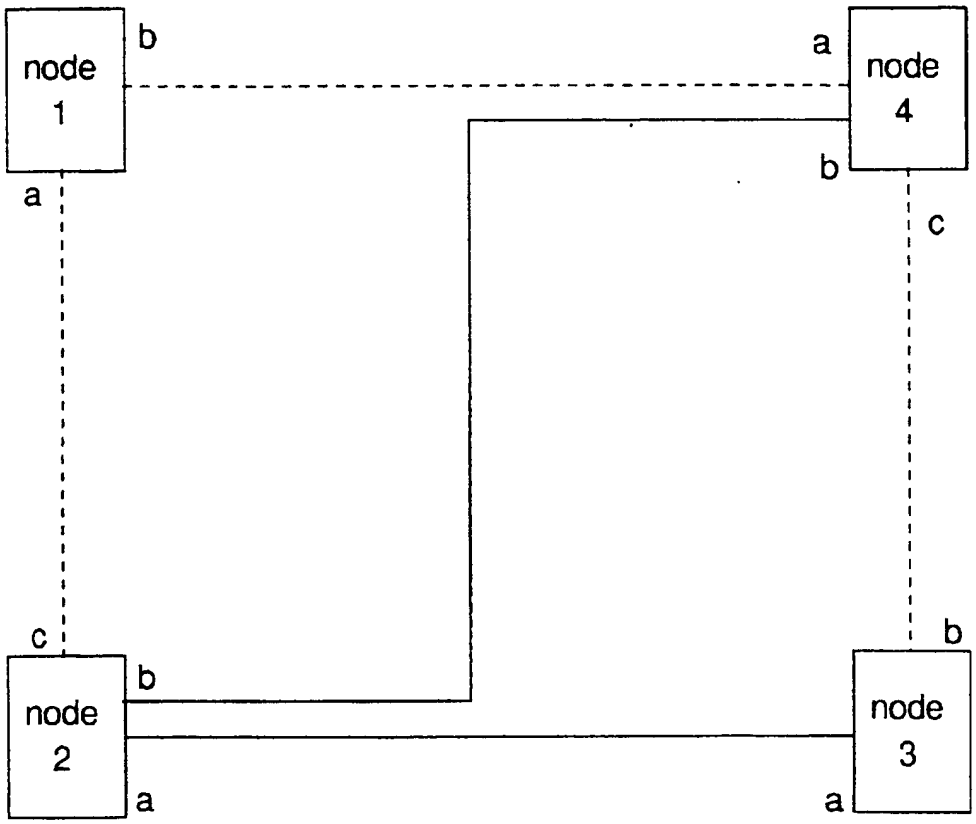


FIG. 3

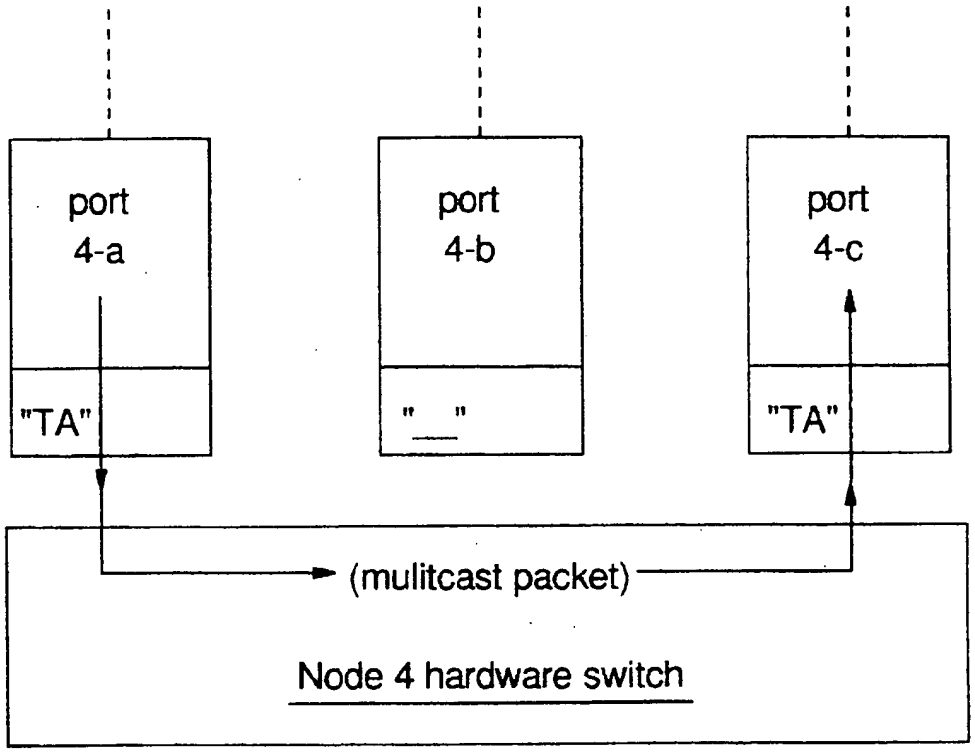

port
4-a
port
4-b
port
4-c
"TA"
" "
"TA"
(mulitcast packet)
Node 4 hardware switch

FIG. 4

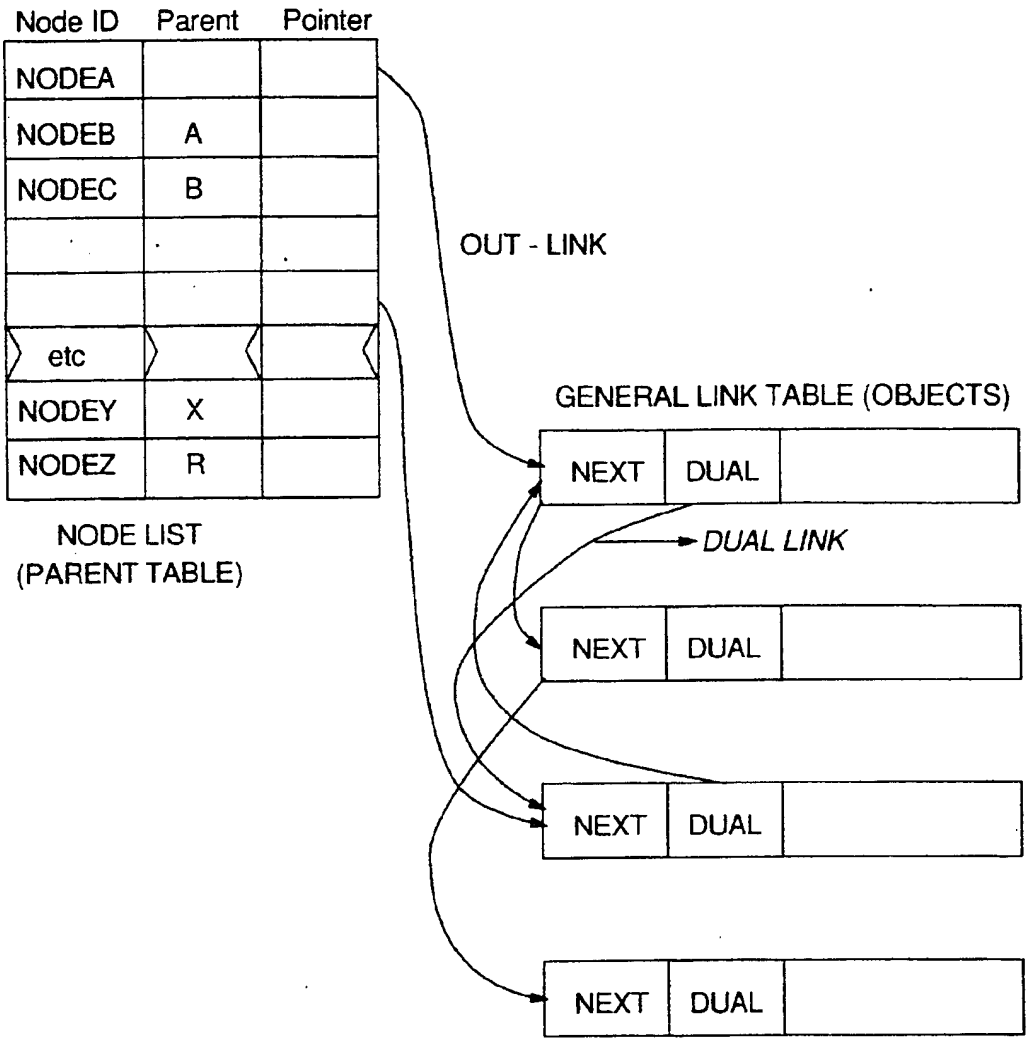

Node ID
Parent
Pointer
NODEA
NODEB
A
NODEC
B
etc
NODEY
X
NODEZ
R
NODE LIST
(PARENT TABLE)
OUT - LINK
GENERAL LINK TABLE (OBJECTS)
NEXT
DUAL
DUAL LINK
NEXT
DUAL
NEXT
DUAL
NEXT
DUAL

FIG. 5

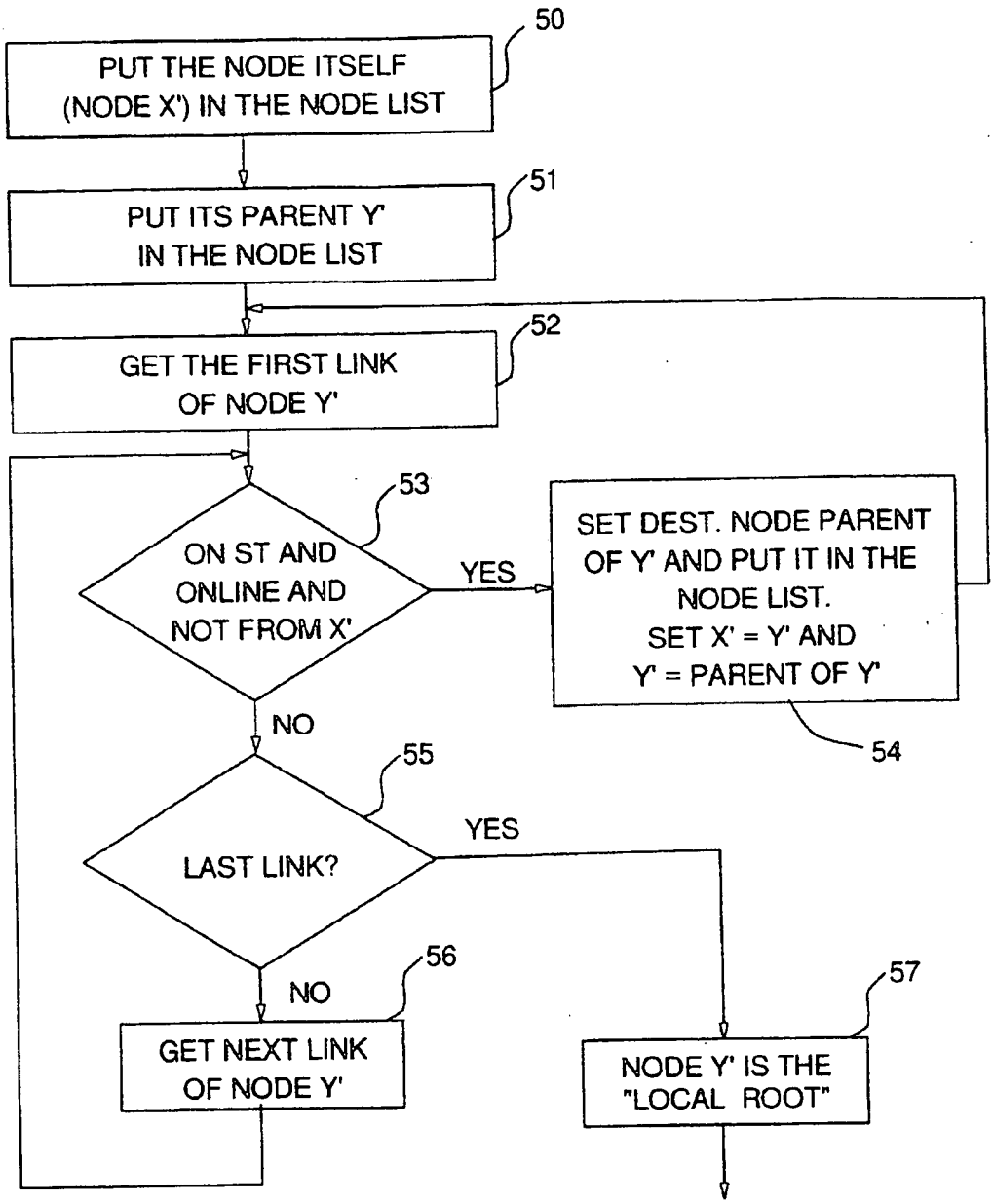

50
PUT THE NODE ITSELF (NODE X') IN THE NODE LIST
51
PUT ITS PARENT Y' IN THE NODE LIST
52
GET THE FIRST LINK OF NODE Y'
53
ON ST AND ONLINE AND NOT FROM X'
YES
SET DEST. NODE PARENT OF Y' AND PUT IT IN THE NODE LIST.
SET X' = Y' AND
Y' = PARENT OF Y'
54
NO
55
LAST LINK?
YES
56
NO
GET NEXT LINK OF NODE Y'
57
NODE Y' IS THE "LOCAL ROOT"

FIG. 6

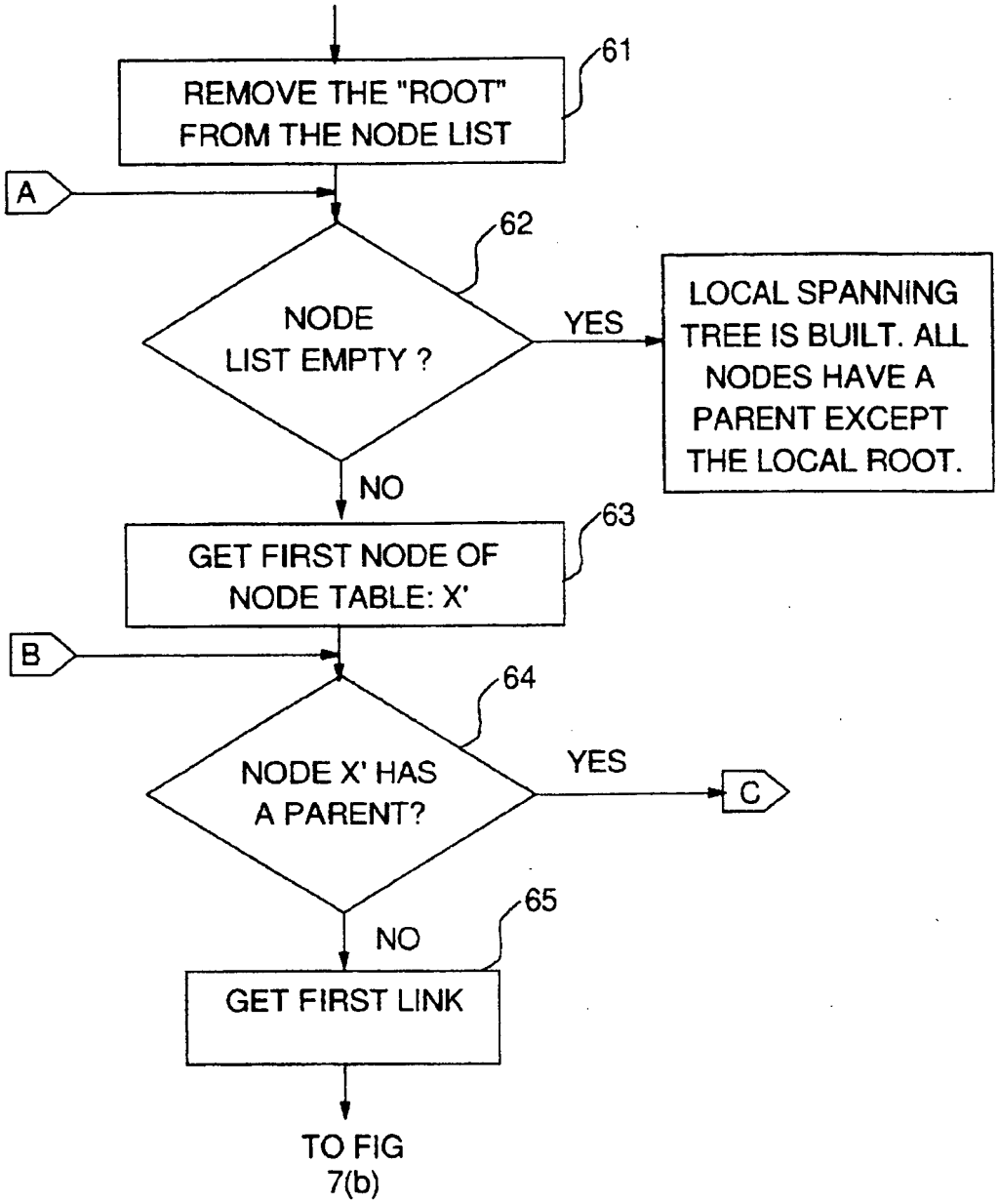

REMOVE THE "ROOT" FROM THE NODE LIST
61
A
62
NODE LIST EMPTY ?
YES
LOCAL SPANNING TREE IS BUILT. ALL NODES HAVE A PARENT EXCEPT THE LOCAL ROOT.
NO
GET FIRST NODE OF NODE TABLE: X'
63
B
64
NODE X' HAS A PARENT?
YES
C
65
NO
GET FIRST LINK
TO FIG 7(b)

FIG. 7(a)

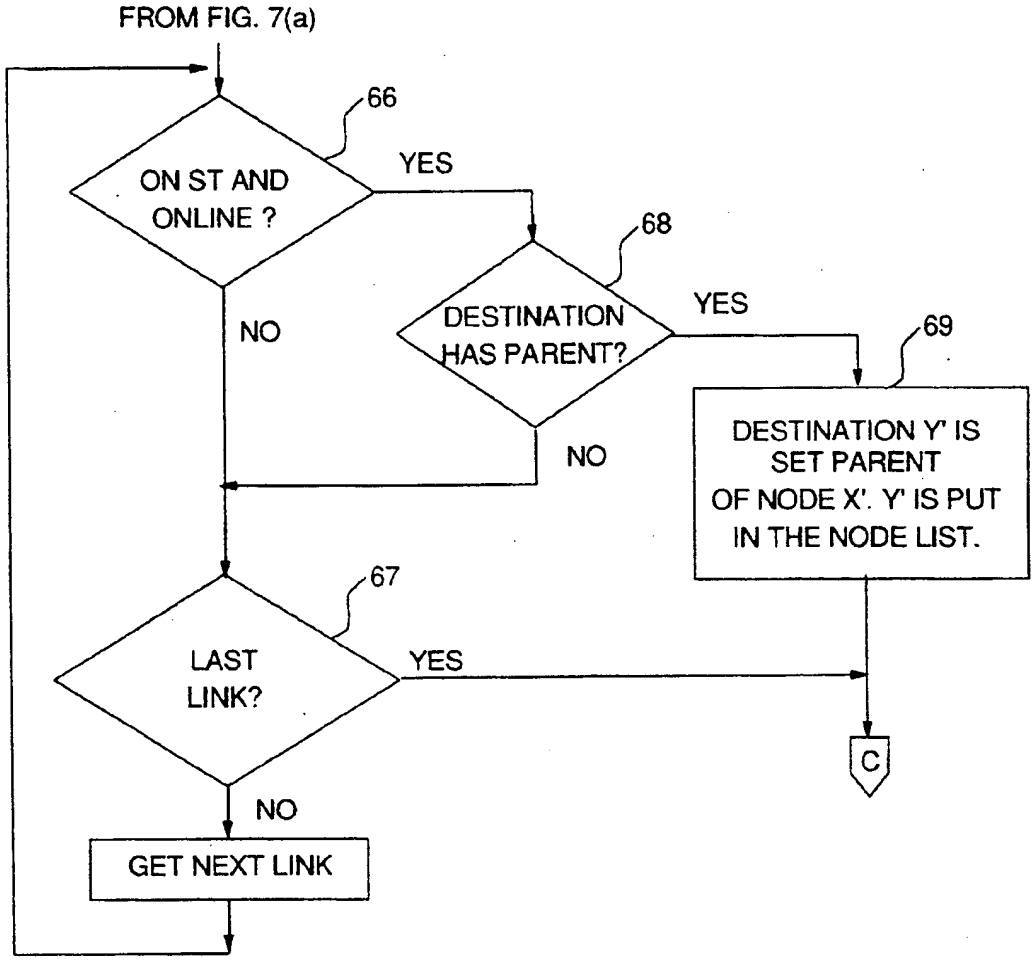

FROM FIG. 7(a)
66
ON ST AND ONLINE ?
YES
NO
68
DESTINATION HAS PARENT?
YES
NO
69
DESTINATION Y' IS SET PARENT OF NODE X'. Y' IS PUT IN THE NODE LIST.
67
LAST LINK?
YES
C
NO
GET NEXT LINK

FIG. 7(b)

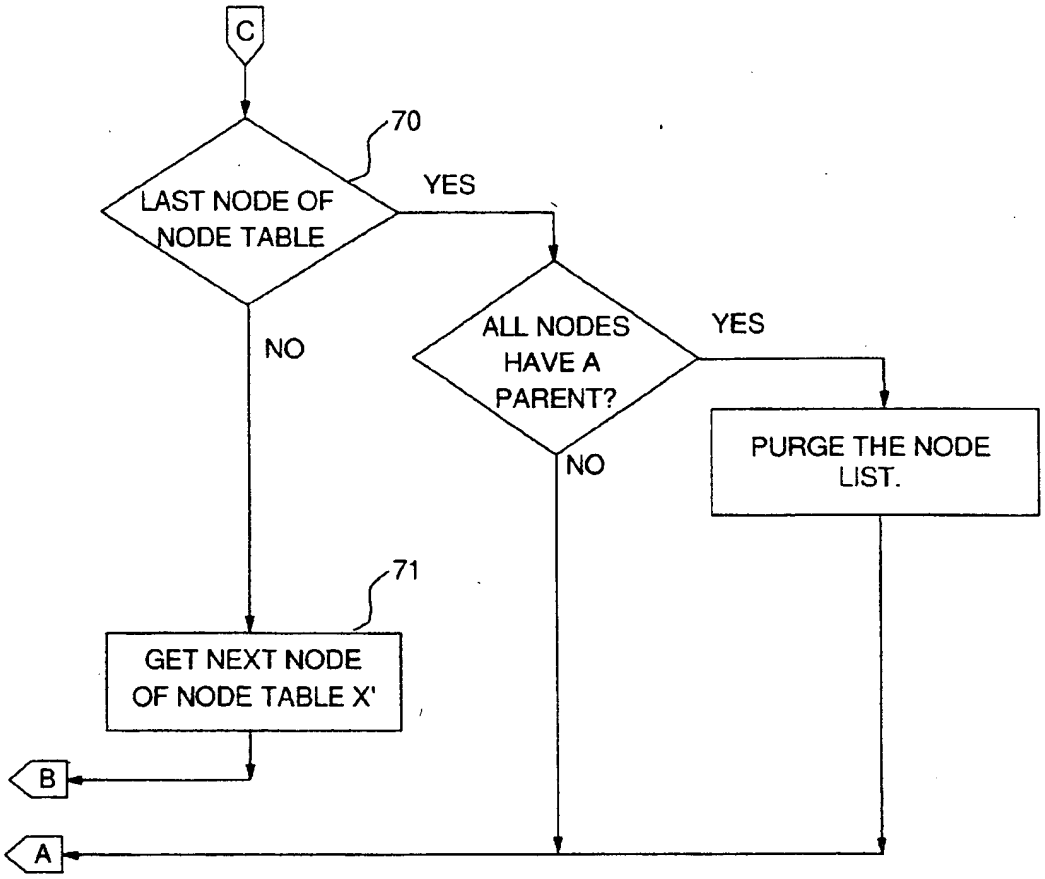

C
70
LAST NODE OF NODE TABLE
YES
NO
ALL NODES HAVE A PARENT?
YES
NO
PURGE THE NODE LIST.
71
GET NEXT NODE OF NODE TABLE X'
B
A

FIG. 7(c)

SYSTEM FOR MANAGING TOPOLOGY OF A NETWORK IN SPANNING TREE DATA STRUCTURE BY MAINTAINING LINK TABLE AND PARENT TABLE IN EACH NETWORK NODE

FIELD OF THE INVENTION

This invention generally deals with data communication networks and more particularly with a so-called Spanning Tree data structure enabling optimized network operation in a number of situations including fast path determination and fast spanning tree recovery. More specifically, the invention deals with a method and means for dynamically organizing each node's spanning tree node data structure to fully reflect the tree topology, as well as a method and means for using the organized node's data structure to implement network operations such as fast path determination or fast spanning tree recovery.

BACKGROUND OF THE INVENTION

FIG. 1 illustrates a data network comprising a number of network nodes A through Z. The network nodes are interconnected by links for the transmission of data packets and other signals, such as control data, between nodes. Each link of the network connects two neighboring nodes. Preferably each link of the network, in fact, comprises a pair of unidirectional links (dual link segments) and each such pair of links can be realized either by a single communication medium, such as an optical fiber, or by two separate communication mediums, such as a pair of coaxial cables or optical fibers.

Such an organized network structure enables data to be communicated between any pair of nodes including a so-called source node and a so-called target node. The source and target designations are dynamic, varying according to the direction of the traffic, which adds to the complexity of the network architecture.

An optimized network organization, which might in some instances be devoted to control traffic, has already been disclosed in the art. One such organization is referred to as a so-called spanning tree architecture.

FIG. 2 depicts a spanning tree, i.e. an optimal structure, that enables the efficient transmission of a message (e.g. data or control) from any node of the communication network to any other node. The spanning tree contains the links needed to interconnect all nodes without any cycles or loops. The links may be considered as extending outward from one node (e.g. node A in FIG. 2) referred to as the root node. The terminating nodes of the spanning tree (e. g. node F, node G, node K, node L, node N, node Z, node T, node W and node Y) are referred to as leaf nodes. The nodes between the leaf nodes and the root node are referred to as intermediate nodes. It should be noted that, in the structure of the network of FIG. 2, any one of the nodes A through Z might be a root node for a specific spanning representation.

In such a tree structure, the root node is designated through a procedure using both hardware and software means distributed within the network. The links connecting pairs of nodes of the spanning tree are sometimes referred to as "edges".

The root node is also referred to as the parent node of the nodes directly connected to it. All nodes connected directly to the root node are referred to as children nodes of the root node. In an outward progression along the tree structure, each node that is directly connected to another node that is closer to the root node is referred to as the child node of the latter (parent) node. Thus, for instance, with the spanning tree structure shown in FIG. 2, node A is the parent of nodes B, H and M; node B is the parent of node C; node C is the parent of nodes D and G. Conversely nodes D and G are both children nodes of node C. In other words, given a spanning tree structure, the root node is the only node with no parent node. Apart from this characteristic, the root node is not necessarily different from any other node of the spanning tree. Consequently, any one of the nodes A through Z may operate as the root node under certain network conditions.

Obviously, under normal operating conditions, several situations may occur that require reorganization of a spanning tree. For instance a new link may be installed for the purpose of optimizing the flow of network traffic (e.g., a low speed or low capacity link may be replaced by a higher speed or higher capacity link) or to add a new path between nodes. Link replacement should, naturally, be implemented as quickly as possible to avoid network traffic degradation between the time the old link is removed and the new link is available.

If a terminal attached to one node requests access to a CPU (Central Processing Unit) attached to another node, the system should simply and quickly select the path between those two nodes. The faster the path is selected, the better in terms of efficient network operation.

The importance of being able to rapidly reorganize a spanning tree in a modern data network becomes more apparent when it is realized any failure (e.g. link failure) will isolate at least a portion of the tree and completely disrupt the traffic therein, if not in the whole data network.

These situations have already been considered in the prior art but the solutions proposed are not considered fully satisfactory in terms of time required for reorganizing a spanning tree.

OBJECTS OF THE INVENTION

One object of the present invention is to provide a data communication network architecture and more particularly a derived spanning tree representation which permits fast spanning tree reorganization under most operating conditions.

Another object of this invention is to provide a node organization and corresponding means for enabling node self-initialization after any tree modification.

Still another object of this invention is to provide a method and means for fast selection of a requested path between two nodes of the spanning tree.

A further object of this invention is to provide a method and means for rapidly reorganizing a spanning tree under a number of operating conditions, including the case of link failure.

Another object of this invention is to provide a method and corresponding means for enabling fast spanning tree recovery even in case of multiple link failures.

SUMMARY OF THE INVENTION

These objects are achieved in a data communication network including several nodes interconnected by bi-directional links for enabling the transmission of data packets and control data between the nodes. The nodes, at any given moment, are organized into a spanning tree including, a root

node and children nodes organized in a parent-children relationship, with each node (except the root node) having a single parent node. Each node is provided with storage means for storing a topology database including a Parent Table, all the nodes of the tree, with each node (except the root node) pointing to its parent node and to a Link Table via out-link pointers, the Link Table including dual link pointers for pointing to respective dual links within the Link Table and means for assigning each link reference with a bit position indicating whether said link currently participates to the spanning tree or not; means for using the Parent Table and Link Table contents for organizing the tree arrangement at will; and means for dynamically updating the node topology database including the Parent Table and the Link Table upon each spanning tree (re)organization.

These and other objects, characteristics and advantages of the invention will become apparent from a consideration of the following detailed description given when read with reference to the accompanying drawings, which specify and show preferred embodiments of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a schematic illustration of a conventional data network wherein the present invention would be implemented.

FIG. **2** is a schematic representation of a spanning tree derived from the data network of FIG. **1**.

FIG. **3** is a schematic representation of a network tree for multicast operation.

FIG. **4** is a schematic representation of the hardware involved in the operation of one of the nodes of the network tree of FIG. **3**.

FIG. **5** is a schematic representation of a node organization which supports the invention.

FIGS. **6** and **7** (including FIG. **7.1** through **7.3**) are flow charts for implementing the invention.

DETAILED DESCRIPTION

The following description assumes a general data network as represented in FIG. **1** wherein, at a given time, certain links might be operating while others are not. A corresponding spanning tree is represented in FIG. **2** with illustrated edges or links connecting the nodes A through Z at the given time.

The network representations of both FIGS. **1** and **2** are provided to illustrate this invention and should in no way be considered as limiting the scope of this invention. The invention may be implemented in networks that are more complex or simpler than that shown in the figures. Also, the distances between nodes may vary from relatively small distances, to relatively large distances (e.g., thousands of kilometers).

It should also be understood that terminals (not represented), including both conventional data terminals and mainframes or CPUs, are attached to given nodes to request and control data traffic using conventional log-on procedures. Those procedures will not be described herein as they do not directly participate in the invention.

Finally, the network nodes and links are not necessarily equivalent to one another in terms of power, speed traffic capacity, etc . . . but this does not interfere with, or affect, the invention.

Also, parent, child and root relationships are defined as described above and the spanning tree representation may, and probably shall, vary over time, with any one of the nodes potentially becoming the spanning tree root.

As already mentioned, data and control information is to be routed throughout the network. Such information, generically referred to as data, includes both pure data and control data, organized into a conventional packet structure. Each packet includes a header section having routing information (addresses) to be used by the network to direct the packet through the network from a source to an assigned target, via network links and intermediate nodes.

Some packets may be broadcast or multicasted to several nodes concurrently. In other words, a multicast routing node enables a packet to be sent from one node to several receiver nodes simultaneously. The sending node and the corresponding receiving nodes act as a multicast tree. One may consider that a tree address is associated with a multicast tree once the tree has been defined. The various network nodes specify which of their associated links (defined in terms of link ownerships) are on the multicast tree by setting the appropriate tree address in the link hardware.

Packets transmitted in the network may contain a tree address in their routing header. Those packets are propagated in the network only on links which are part of the multicast tree and have the right tree address set in their hardware.

An example of multicast tree organization and multicast routing is illustrated in both FIGS. **3** and **4**. Represented in FIG. **3** is a network including four nodes labeled nodes **1**, **2**, **3** and **4** respectively. Those nodes are interconnected by physical links, each link consisting of two unidirectional links. Each unidirectional link is defined by (one may say "belongs to") its origin node. Conversely, the node is said to "own" links. Accordingly: Node **1** owns two links—links **1**-*a* and **1**-*b*; Node **2** owns three links—links **2**-*a*, **2**-*b* and **2**-*c*; Node **3** owns two links—links **3**-*a* and **3**-*b*; and Node **4** owns three links—links **4**-*a*, **4**-*b* and **4**-*c*.

As represented in FIG. **3**, the link **4**-*a* is said to be the dual link of link **1**-*b*, with both links forming a complete bidirectional physical link between Node **1** and Node **4**.

One possible multicast tree connects the four nodes through the dotted links. This multicast tree is identified by a preassigned tree address "TA". When node **1** sends a packet with the address TA in its routing header section, the packet is automatically sent on link **1**-*a* and link **1**-*b*, then routed from node **4** to node **3** via link **4**-*c*. Only those links that are part of the multicast tree will propagate the packet. The packet from node **1** is said to have been broadcast to nodes **2**, **3** and **4** via the multicast tree as required.

FIG. **4** is a schematic representation of switching hardware supporting the multicast tree "TA" in node **4**. The node **4** is provided with a number of ports corresponding to each link attached to node **4**, i.e. port **4**-*a*, port **4**-*b* and port **4**-*c*. Each port is provided with a buffer for storing the multicast tree address. A packet is multicasted only through those ports (excluding the input port) having a stored multicast tree address matched by a corresponding tree address in a packet header.

When a packet with a "TA" field in its routing header is received in node **4**, the node's internal switch hardware copies the packet to all links marked as being part of the "TA" multicast tree, except the link on which the packet arrived on the node. In the represented case, the packet is forwarded to link **4**-*c*.

A Spanning Tree (ST)is an example of a multicast tree that joins all the nodes in a network (see FIGS. **1** and **2**). Any

packet sent by a node and carrying the Spanning Tree address in its routing header section, shall automatically be sent to all other nodes of the network by using the above described multicasting process and hardware.

This mechanism is particularly useful for propagating network control messages that must reach quickly all network nodes without being transported on each and every link of the original data communication network, which would result in the message being received more than once by many of the nodes. The spanning tree insures that a message sent by a sender node on the spanning tree is received only once by each receiver node belonging to the spanning tree.

In the present invention, each node includes means for storing a full spanning tree representation or topology database and means for dynamically readjusting this full topology representation in a very fast manner. To that end, each node is provided with software means, which, using conventional and preexisting processor means and random access memory means, as well as hardware and logic of FIG. 3 and 4 as already described, to organize and maintain the spanning tree representation to enable optimal operation of the network.

More precisely, each node is provided with Control Point Adapter logic including control unit and random access memory means wherein a topology database will be stored. The topology database shall list the network nodes, with each node entry designating a parent node and including an out-link pointer pointing to an outlink table. The outlink table will, in turn, include dual link pointers pointing to a dual link table as well as a bit position (ST bit) indicating whether the considered link is, actually, on the spanning tree (ST) or not.

For instance, consider Node C. Assuming the network topology shown in FIG. 1, Node C's topology database will include, as does each other node, a parent table (node list) stored therein. The outlink table will list in node list form, all network nodes. Each network node entry in the table will include a pointer to the parent node for the network node. Each network node entry will also point to an out-link table listing the node links. For instance, Node C shall point to a list including CB, CD, CG and CI (those same links are linked together with CB connecting to CD, CD connecting to CG and CG connecting to CI.

The out-link table also includes indications as to whether the listed links belong to the actual spanning tree or not. As mentioned a single bit (ST) is sufficient to provide this indication. If an ST bit is set to a binary value "1", this indicates that the associated link belongs to the spanning tree. If ST equals "0", the link is off the tree. For example, given the spanning tree representation of FIG. 2, ST=1 for links CB, CD and CG, and ST=0 for CI.

Each link of the out-links table points to a second or dual link table identifying the outbound link from the node at the opposite end of the specified link. For instance, CB shall point to a list including BC, BI, BA and BD. Obviously, due to the duality context, BC in the second table shall include a pointer pointing back to CB in the first list. The second list also mentions whether each link on the second list is on the spanning tree or not. ST bit is accordingly set to "1" in front of BC and BA, while it is at the binary value zero in front of BI and BD.

Each link entry within said second list points to a third dual list of links from nodes at the opposite ends of each link on the second list. For instance BA shall point to a third list including AD, AB, AI, AH, AM, AO and AQ. The following links in said third list will show an ST bit set to the binary value 1, i.e. AB, AH and AM, while ST=0 for the remaining links.

The hierarchical link listing will expand until the network has been entirely described. (All these link tables or lists may, in practice be combined into a single LINK TABLE). The parent table contents are node dependent as the table represents a view as seen from a particular node, but also each table includes a full tree representation including each and every node's parent. Those tables might in fact be combined into a topology database schematically represented in FIG. 5.

FIGS. 6 and 7 are flow charts of processes for building a local Spanning Tree representation. FIG. 6 describes how to determine the root of the local image of the Spanning Tree. The local root is the first terminal node (node with no other child node) encountered when selecting successive parents.

Assume the process is started at a given node X'. The process starts with step 50 whereby the given node X' is itself entered in the node list. Then, in step 51, a corresponding parent node Y' is entered into the node list. Using the existing topology database and network spanning tree the system identifies first link of Node Y' (step 52) and performs a check (step 53) whether the identified link is on the Spanning Tree and is on-line and does not come from node X'. If the result of the tests of step 53 is positive, the node at the opposite end of the identified link is identified as a parent of Y' and this status is recorded in the node list. X' is set equal to Y' and new node is made parent of Y' (step 54). The process then loops back to step 52.

Should, on the contrary, the result of the test of step 53 be negative, a second test is performed (step 55) to determine whether the link under consideration was the last link for the node. If more links to the node remain, the system then considers next link of node Y' (step 56). Otherwise, the process proceeds to step 57 and Node Y' is recorded as being the local root.

The process then goes to FIG. 7 representing the algorithm for scanning the node table and database to update the corresponding parent information. The root node is first removed from the node list (step 61), and a test is performed to check whether the node list is empty (step 62). Should this be the case, the local spanning tree is fully formed, with all nodes, but the local root, having a parent node defined and recorded. Otherwise, the process reads the first node of the node table (e.g. Node X') (step 63) and a test is performed to check whether this node X' has a parent node (step 64). Should the answer be negative, the process identifies a first link to node X' (step 65) and tests whether this link is on the spanning tree (ST=1) and is currently on-line or available (step 66). If the answer to step 66 is negative, a check is performed (step 67) to see whether the link under consideration is the last link to node X' to be considered. If this is not the case, the next link is selected and the process loops back to step 66. In case of a positive answer at step 66, the system examines whether the node at the other end of the link under consideration has a parent (step 68) and if not, the system looks for another link in step 67.

Otherwise, the destination node (say Y') is set as being the parent of node X', and Y' is recorded in the node list (step 69). Should the results of either of tests 64 and 67 be positive, and upon completion of the operations of step 69 as well, another test (step 70) is performed to check whether the considered node was the last node to be scanned in the node table.

In case of negative answer to this last test, the system scans the node table to identify its next recorded node (step

71) and the process loops back to step 64. Otherwise, the system checks whether all nodes have a parent (step 72), and if not, loops back to step 62, if all nodes have an identified parent, otherwise, the node list is purged and the process loops back to the same step 62.

As already mentioned, the full local tree representation provided by this invention improves the network and spanning tree operations in a number of critical situations, by greatly simplifying and speeding-up the resolution of any corresponding problem.

Consider the case where a path is to be quickly determined between a source node and a target node, using only the spanning tree edges (links). A fast path determination is initiated by the system. This operation is performed by exploring the source node topology database, up to the root node, for both the source node and the target node, listing all target's parent nodes up to the root node and all source's parent nodes up to the root node. The first common node on the two lists defines the path.

As an example, assume the source node Z wishes to establish a spanning tree path to target node X.

Referring to the spanning tree shown in FIG. 2, the logic set within node Z will scan the local Table and establish the following lists (see FIG. 2):

source-parent-list=Z, R, Q, P, O, M, A.

target-parent-list=X, U, R, Q, P, O, M, A.

Then the system proceeds with a reverse scanning of both lists above, deleting all common nodes but the last. Consequently, on both lists, A is first deleted, then M, O, P and Q. The last common node being R, Node Z will set the path looked for and define it in a control message, as being Z, R, U, X by locally scanning the remaining source list forward and target list backward.

But the present invention is even more important in situations wherein a link becomes unavailable and the spanning tree must be redefined very quickly. With the full tree representation provided by this invention in topology databases at each node, the system can recover quickly from edge (link) failures. The distributed tree maintenance algorithm disclosed above defines a unique node called the root. The root centralizes and coordinates most of the maintenance of the spanning tree.

When an edge failure occurs, the original spanning tree is partitioned into two trees. A node that was a child before the failure may now become a root (node with no parent) of a newly created tree partition. With the information and systems already existing at each node,the nodes in each partition may try to define a spanning tree for their partition. Ultimately, any spanning trees for the two partitions should be merged into a unique single tree again bypassing the failed link. In order for the two partition spanning trees to merge, their respective roots must agree on a single edge at which they should be interconnected. A method for updating a spanning tree after network topology changes (link and node failures and recoveries) was described by Israel Cidon, Inder S. Gopal, Shay Kutten and Marcia Peters, in the IBM Technical Disclosure Bulletin Vol 35 No 1A, June 1992, pages 93–398, in an article having the title "Distributed Tree Maintenance". In that article, the topology maintenance algorithm ensures that nodes on a tree eventually know the topology of their neighboring edges. Agreement between root nodes for interconnecting is ensured through a property called "weight" assigned to all network edges in a unique manner, such that they can be strictly ordered. For two trees to merge, each tree's root must agree on the non-tree edge between them with the lowest weight. Then the root moves from one node to another via a reliable point to point message called Move-Root. The only node that can send Move-Root is the current root. Upon sending the Move-Root message, the current root is no longer the root. The receiver of Move-Root becomes the next root. The topology update and tree maintenance algorithms are interdependent, in that the current root considers all the edges in the network to determine where to move the root.

The full knowledge of tree topology, and more particularly the knowledge of node's parentships in each node as provided by this invention combined with the fast path determination derived therefrom, as described above, enables faster tree recovery. This process, named fast spanning tree recovery to be initiated by the system shall be described hereunder and illustrated in one example.

Assume the edge BC fails. The original tree (see FIG. 2) is now partitioned into two trees, the original one (Tree (A)) excluding nodes C, G, D, E and F, and a new tree including only those nodes C, G, D, E and F. By definition, the child node at the failing edge, that is node C, is assumed to be the root for the tree including nodes C, D, E, F, and G. This second tree may be labelled Tree (C).

When the process for merging the two partitions, i.e. Tree (A) and Tree (C) starts with Node C scanning its topology database. The database is represented schematically hereunder and shows the Parent Table and the Link Tables which, in fact, could be easily combined into a single Link Table.

The Parent Table at Node C lists all the network nodes and, for each node, the corresponding parent node. Thus, prior to the link failure, Node A, being the root, is the only node without a parent. Node B's parent is A, Node C's parent is B, and so-on down to node Z. The Parent Table is therefore as follows prior to the link failure:

PARENT TABLE

Node	Parent	Out-link Pointer
Node A	Void	
Node B	A	
Node C	B	→ to Link Tables
:	:	
Node X	U	
Node Y	X	
Node Z	R	

As indicated, the Parent Table also includes an out-link pointer for each listed node, the pointer pointing to the links connected to the listed node (first list), with each link list in turn pointing to a dual link list (second list, third list, etc . . .)

Consider one example with a first list showing the out-links from node C, a second list showing the dual list pointed to by CB dual PTR, and a third list pointed to by BA dual PTR.

LINK TABLE

First List	Links	Links Pointers	Dual PTR	ST bit
	CB			1
	CD			1
	CG			1
	CI			0

-continued

LINK TABLE

Second List		ST bit
	BC	1
	BI	0
	BA	1
	BD	0

Third List		ST bit
	AD	0
	AB	1
	AI	0
	AH	1
	AM	1
	AQ	0

As shown above, the out-link pointer in front of Node C reference, points to the first link list listing including CB, CD, CG and CI, each pointing to the next link (not shown in the above tables). In other words, CB points to CD which in turn points to CG, and so-on until CI. In addition and as mentioned, the first link list includes dual-link pointers pointing to the second link list. For instance, the dual pointer associated with CB in the first link list points to the dual list (or second link list) listing the following links: BC, BI, BA and BD. Each link reference in said second Link List, in turn points third link list, and so on, as shown for link BA dual pointer. All these link lists are, in practice, combined into a single complex Link Table.

In addition, and this should help in the link failure recovery as mentioned, each link is defined with a one bit reference (i.e. ST) indicating whether the corresponding link is currently on the spanning tree (ST=1), or off the spanning tree (ST=0). For instance, for the original tree as represented in FIG. 2, and for Node C the first Link List shall indicate ST=1 CB, CD, CG and ST=0 for CI.

Once the link BC/CB fails, as mentioned above, Node C, i.e. the child node on the failing edge, becomes a root node for the tree including C, G, D, E and F. Only those links on Tree(C), i.e. CG, CD, DE, EF keep an ST bit equal to binary value 1. The others are reset. The above first table then looks as follows

Link	ST
CB	0
CD	1
CG	1
CI	0

The Parent Table is also reset to reflect the link failure and tree partitioning. Given the above information, just by scanning its topology database, and particularly by scanning the Link Table, Node C shall determine the list of links XKn, where Kn is a node belonging to Tree (A). In the example above the list of links CKn, shall only include CB and CI as links not belonging to Tree (C) and therefore potentially usable for interconnecting both Tree (A) and Tree (C).

Since CB is failing, the list of remaining links includes only CI and CI shall be designated for selection. More generally speaking, the list of CKn might include several links. In this case, an additional parameter would be predefined (e.g. highest speed link) to resolve the ambiguity and select the most appropriate link from multiple possible links.

Consequently the link selected by Node C is CI. Node C starts a dialog with node I by sending a so-called "Combine Request" message. Node I, can, however, respond only if it is the new root for what had been Tree (A). To avoid waiting indefinitely for node I to become the root, in case a problem occurs. Node C sets a timer and waits only for the duration of the set time.

Using a similar process (Table resettings, etc . . .) on Tree A, original root node A also selects a link to be used to connect the two partitioned trees. In other words, once receiving the message that link BC failed, the original root (A)scans its topology database to find the list of candidate links KnC where Kn is a node belonging to Tree (A) after the tree split occurred, i.e. the tree including all the network nodes but C, G, D, E, F.

The same link IC among the set of CKn should be selected by both the root nodes A and C in the two partitions.

An immediate Move Root may then be implemented from node A to node I by using the fast path determination procedure of this invention as described above (just through parent lists scanning). Node A issues a Move-Root message to node I using the selected path on the spanning tree between node A and node I. Once node I becomes the new root, it can respond to the "Combine Request" message received from Node C over CI and complete the merge process with C. Then the unified spanning tree is redefined according to the process described above, with Node I being the new tree root.

As already mentioned, the present invention should appear to be of particular interest in a number of other situations. For instance to reduce a given spanning tree diameter, or to replace a given low grade (or so-called low weight) link by a higher weighted link. In those instances, the full topology databases available in each node shall appear fairly convenient.

We claim:

1. In a data communication network having a plurality of nodes and a plurality of bi-directional links, each of said nodes being connected to at least one other node through one of said bi-directional links to enable data to be exchanged between the nodes, said network being defined by at least one spanning tree comprising a root node and one or more children nodes, each of said children nodes except the root node having a single parent node, a topology manager located in at least one of said nodes comprising:

a) a link table memory containing a link table for each node in the current spanning tree, each link table identifying the node with which the link table is associated and each of the links from the identified node to other nodes in the network, each link entry in a link table including a spanning tree entry which indicates whether the link is currently included in the current spanning tree,

b) a parent table memory containing a node entry for each node in the current spanning tree, each node entry other than the entry for the root node further containing the address of the node's parent node, the address of the link table for the node and the identification of dual links which are common to the link table for the node and link tables for other nodes; and

c) spanning tree update logic for updating the link tables and the parent table upon a change in network topology

affecting the current spanning tree, said spanning tree update logic further including

i) means for scanning the parent table memory to detect an entry identifying a first node for which no parent node is identified, said means assigning the root position in a new spanning tree to the first node,

ii) means for reading the parent table and link table for the first node to identify at least one adjacent node which satisfies predetermined criteria for a parent node to the first node, marking the adjacent node as a parent to the first node and amending the parent tables and link tables accordingly, and

iii) means for repeating the operations performed by said reading means with each identified adjacent node being assigned the role as a first node until all nodes have a parent and amending the parent tables and link tables accordingly.

2. A topology manager as defined in claim 1 further including route selection means for quickly determining a path between any two nodes on the spanning tree, one of said nodes being identified as a source node and the other being identified as a target node, said route selection means being located at the source node and comprising:

a) means for scanning the parent table memory to create a first list comprising nodes on the spanning tree beginning with the source node and ending with the spanning tree root node and a second list comprising nodes on the spanning tree beginning with the target node and ending with the spanning tree root node;

b) means for processing the first and second lists by eliminating the spanning tree root node from both lists and each succeeding child node until all nodes common to both lists have been eliminated except the common node closest to the source node on the first list and to the target node on the second list; and

c) means for defining the route between the source node and the target node as the concatenation of the nodes remaining on the first and second lists.

3. In a data communication network having a plurality of nodes and a plurality of bi-directional links, each of said nodes being connected to at least one other node through one of said bi-directional links to enable data to be exchanged between the nodes, said network being defined by at least one spanning tree comprising a root node and one or more children nodes, each of said children nodes except the root node having a single parent node, a topology manager located in at least one of said nodes comprising:

a) a link table memory containing a link table for each node in the current spanning tree, each link table identifying the node with which the link table is associated and each of the links from the identified node to other nodes in the network, each link entry in a link table including a spanning tree entry which indicates whether the link is currently included in the current spanning tree,

b) a parent table memory containing a node entry for each node in the current spanning tree, each node entry other than the entry for the root node further containing the address of the node's parent node, the address of the link table for the node and the identification of dual links which are common to the link table for the node and link tables for other nodes;

c) spanning tree update logic for updating the link tables and the parent table upon a change in network topology affecting the current spanning tree; and

d) route selection means for quickly determining a path between any two nodes on the spanning tree, one of said nodes being identified as a source node and the other being identified as a target node, said route selection means being located at the source node and comprising

i) means for scanning the parent table memory to create a first list comprising nodes on the spanning tree beginning with the source node and ending with the spanning tree root node and a second list comprising nodes on the spanning tree beginning with the target node and ending with the spanning tree root node,

ii) means for processing the first and second lists by eliminating the spanning tree root node from both lists and each succeeding child node until all nodes common to both lists have been eliminated except the common node closest to the source node on the first list and to the target node on the second list, and

iii) means for defining the route between the source node and the target node as the concatenation of the nodes remaining on the first and second lists.

4. A topology manager as defined in either of claims 1 or 3 further including fast recovery logic for recovering from network fragmentation resulting from a failure of a link forming part of an original spanning tree, the fragmentation resulting in a first set of nodes including an orphaned node which had been connected to the original spanning tree through the failed node and all nodes connected to that node directly or through intervening nodes and a second set of nodes including the original root node and all nodes still connected to the original root node directly or through intervening nodes, said fast recovery logic including:

a) means at one or more of the nodes in said first set of nodes for designating the orphaned node as the new root node for the first set of nodes;

b) means in both the new root node and the original root node for resetting the link tables to define a first spanning tree for the first set of nodes and a second spanning tree for the second set of nodes;

c) means in both the original root node and the new root node for identifying each remaining link between nodes in the first spanning tree and nodes in the second spanning tree and for selecting one of the remaining links to provide the new connecting link between the first spanning tree and the second spanning tree;

d) means for designating the node connected to the selected link in the first set of nodes as a new parent node through which the first set of nodes are to be connected to the second set of nodes.

5. For use in a data communication network having a plurality of nodes and a plurality of bi-directional links, each of said nodes being connected to at least one other node through one of said bi-directional links to enable data to be exchanged between the nodes, said network being defined by at least one spanning tree comprising a root node and one or more children nodes, each of said children nodes except the root node having a single parent node, a method of managing the topology of the network, said method being implemented in at least one node and comprising the steps of:

a) maintaining a link table for each node in the current spanning tree, each link table identifying the node with which the link table is associated and each of the links from the identified node to other nodes in the network, each link entry in a link table including a spanning tree entry which indicates whether the link is currently included in the current spanning tree,

b) maintaining a parent table containing a node entry for each node in the current spanning tree, each node entry

other than the entry for the root node further containing the address of the node's parent node, the address of the link table for the node and the identification of dual links which are common to the link table for the node and link tables for other nodes;

c) updating the link tables and the parent table upon a change in network topology affecting the current spanning tree; and

d) quickly selecting a route between any two nodes on the spanning tree, one of said nodes being identified as a source node and the other being identified as a target node, said route selection steps comprising

i) creating a first list comprising nodes on the spanning tree beginning with the source node and ending with the spanning tree root node and a second list comprising nodes on the spanning tree beginning with the target node and ending with the spanning tree root node,

ii) beginning with the spanning tree node, eliminating the spanning tree root node from both lists and each succeeding child node until all nodes common to both lists have been eliminated except the common node closest to the source node on the first list and to the target node on the second list, and

iii) concatenating the nodes remaining on the first and second lists to define the route between the source node and the target nodes.

6. For use in a data communication network having a plurality of nodes and a plurality of bi-directional links, each of said nodes being connected to at least one other node through one of said bi-directional links to enable data to be exchanged between the nodes, said network being defined by at least one spanning tree comprising a root node and one or more children nodes, each of said children nodes except the root node having a single parent node, a method of managing the topology of the network, said method being implemented in at least one node and comprising the steps of:

a) maintaining a link table for each node in the current spanning tree, each link table identifying the node with which the link table is associated and each of the links from the identified node to other nodes in the network, each link entry in a link table including a spanning tree entry which indicates whether the link is currently included in the current spanning tree,

b) maintaining a parent table containing a node entry for each node in the current spanning tree, each node entry other than the entry for the root node further containing the address of the node's parent node, the address of the link table for the node and the identification of dual links which are common to the link table for the node and link tables for other nodes; and

c) updating the link tables and the parent table upon a change in network topology affecting the current spanning tree, said updating step further including the steps of

i) scanning the parent table memory to detect an entry identifying a first node for which no parent node is identified and assigning the root position in a new spanning tree to the first node,

ii) reading the parent table and link table for the first node to identify at least one adjacent node which satisfies predetermined criteria for a parent node to the first node, marking the adjacent node as a parent to the first node and amending the parent tables and link tables accordingly, and

iii) repeating the operations performed by said reading means with each identified adjacent node being assigned the role as a first node until all nodes have a parent and amending the parent tables and link tables accordingly.

7. A method of managing network topology as defined in claim 6 further including steps of quickly selecting a route between any two nodes on the spanning tree, one of said nodes being identified as a source node and the other being identified as a target node, said route selection steps comprising:

a) creating a first list comprising nodes on the spanning tree beginning with the source node and ending with the spanning tree root node and a second list comprising nodes on the spanning tree beginning with the target node and ending with the spanning tree root node;

b) beginning with the spanning tree node, eliminating the spanning tree root node from both lists and each succeeding child node until all nodes common to both lists have been eliminated except the common node closest to the source node on the first list and to the target node on the second list; and

c) concatenating the nodes remaining on the first and second lists to define the route between the source node and the target nodes.

8. A topology manager as defined in either of claims 6 or 5 further including steps for recovering from network fragmentation resulting from a failure of a link forming part of an original spanning tree, the fragmentation resulting in a first set of nodes including an orphaned node which had been connected to the original spanning tree through the failed link and all nodes connected to that node directly or through intervening nodes and a second set of nodes including the original root node and all nodes still connected to the original root node directly or through intervening nodes, said steps including:

a) resetting the link tables to define a first spanning tree for the first set of nodes and a second spanning tree for the second set of nodes;

b) identifying each remaining link between nodes in the first spanning tree and nodes in the second spanning tree;

c) selecting one of the remaining links to serve as a new connecting link between the first spanning tree and the second spanning tree; and

d) designating the node connected to the selected link in the first set of nodes as a new parent node through which the first set of nodes are to be connected to the second set of nodes in a new spanning tree including both said first set of nodes and said second set of nodes.

* * * * *

US005627766A

United States Patent [19]

Beaven

[11] Patent Number: 5,627,766

[45] Date of Patent: May 6, 1997

[54] **PERFORMANCE AND STATUS MONITORING IN A COMPUTER NETWORK**

[75] Inventor: **Paul A. Beaven**, Romsey, Great Britain

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **368,075**

[22] Filed: **Jan. 3, 1995**

[30] **Foreign Application Priority Data**

Feb. 8, 1994 [GB] United Kingdom 9402380

[51] **Int. Cl.**[6] **G06F 11/34**

[52] **U.S. Cl.** ..... **364/551.01**; 364/514 B; 364/550; 370/241; 395/200.11

[58] **Field of Search** ..... 364/514 B, 550, 364/551.01; 370/13, 17; 395/200.11

[56] **References Cited**

FOREIGN PATENT DOCUMENTS

0510822 10/1992 European Pat. Off. .

OTHER PUBLICATIONS

IBM, "MQ Series", Message Queue Interface Technical Reference (SC33-0850-01), Third edtition. Nov. 1994.

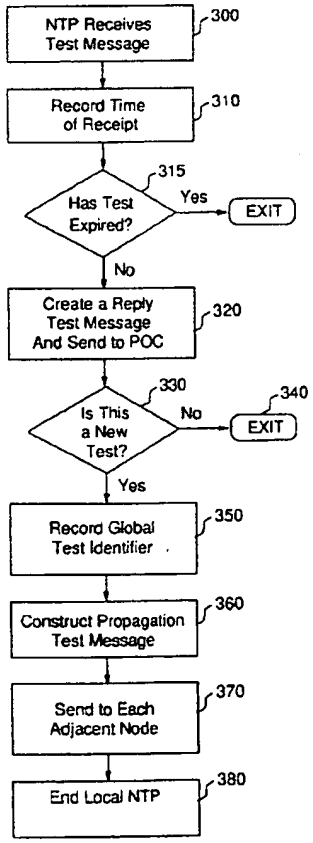
Primary Examiner—Edward R. Cosimano

Attorney, Agent, or Firm—John J. Timar

[57] **ABSTRACT**

Provided is a method and a system for computer network monitoring, implemented in a network in which processes communicate using message queuing. Each node of the network has a network management program installed thereon which includes two independent components: a Point Of Control (POC) program for initiating network tests by injecting a test message into the network and for receiving responses from all the nodes of the network; and a Network Test Program (NTP) for sending a reply message to the single POC for a particular test when the NTP receives test messages within that test, and for propagating the test by forwarding a message to all of the current node's adjacent nodes. Test results are analyzed at the POC for display to the network administrator. Injected test messages propagate throughout the network in a self-exploring manner, exploiting the parallelism of the network. The individual nodes are not required to know the network topology other than to know their nearest neighbor nodes.

18 Claims, 3 Drawing Sheets



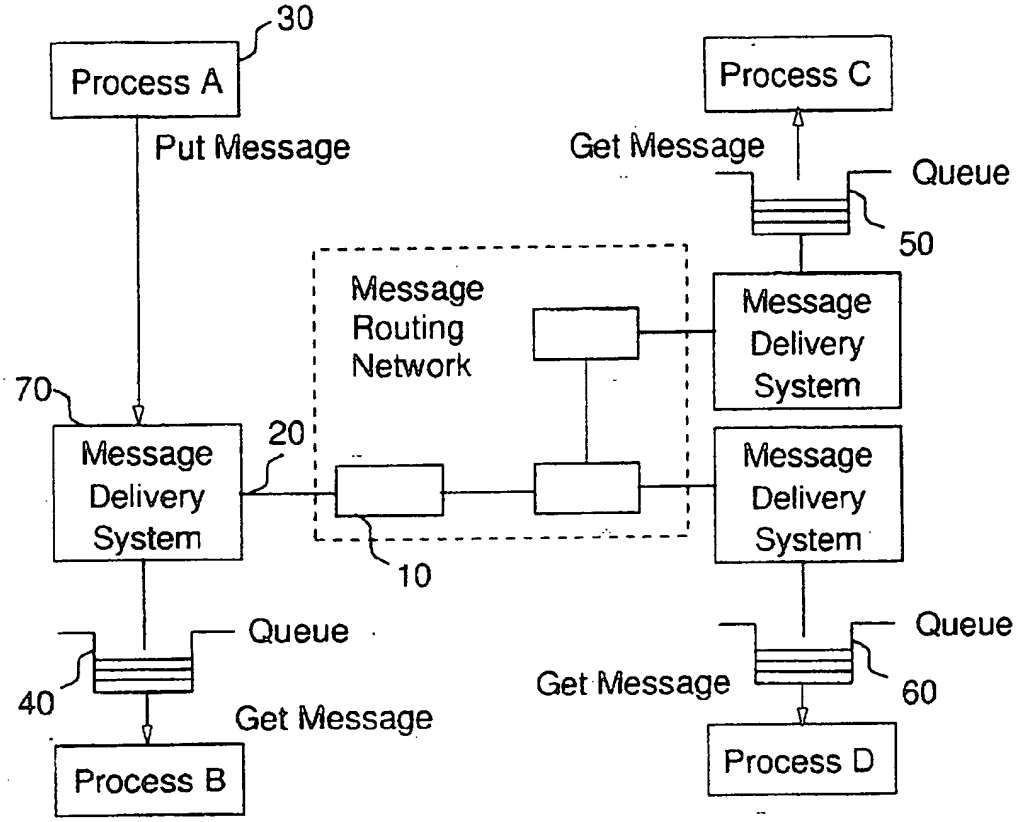

30
Process A
Put Message
Process C
Get Message
Queue
50
Message
Routing
Network
Message
Delivery
System
70
Message
Delivery
System
20
Message
Delivery
System
10
Queue
40
Get Message
Process B
Queue
Get Message
60
Process D

FIG. 1

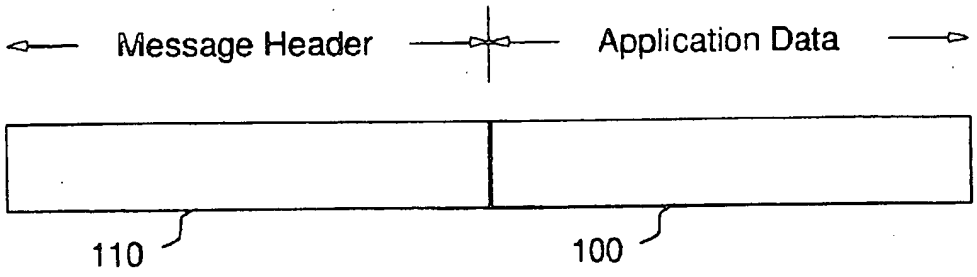

Message Header
Application Data
110
100

FIG. 2

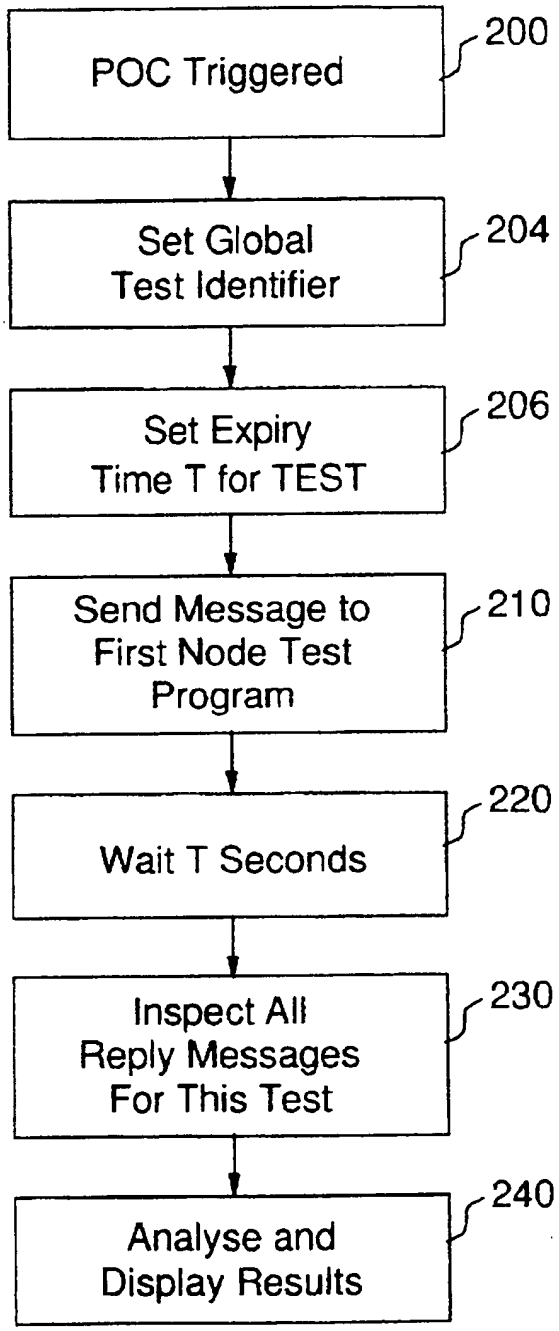

POC Triggered
200
Set Global Test Identifier
204
Set Expiry Time T for TEST
206
Send Message to First Node Test Program
210
Wait T Seconds
220
Inspect All Reply Messages For This Test
230
Analyse and Display Results
240

FIG. 3

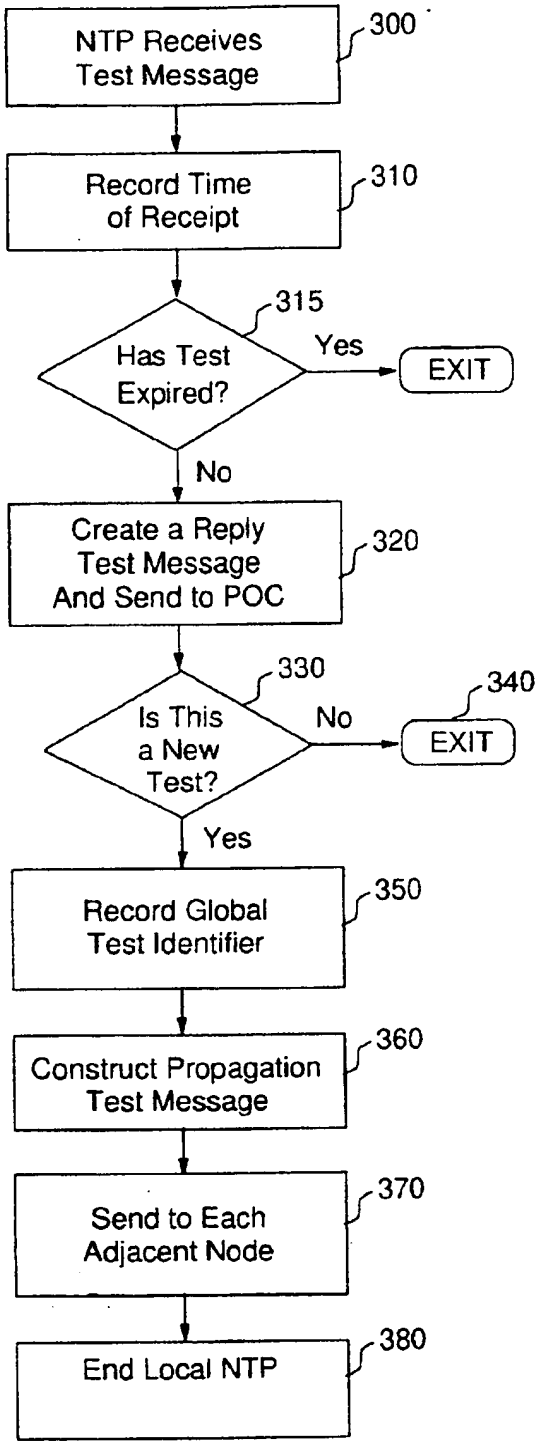

NTP Receives Test Message
300
Record Time of Receipt
310
315
Has Test Expired?
Yes
EXIT
No
Create a Reply Test Message And Send to POC
320
330
Is This a New Test?
No
340
EXIT
Yes
Record Global Test Identifier
350
Construct Propagation Test Message
360
Send to Each Adjacent Node
370
End Local NTP
380

FIG. 4

PERFORMANCE AND STATUS MONITORING IN A COMPUTER NETWORK

FIELD OF INVENTION

The present invention relates to computer networks and more particularly to monitoring the nodes and communications links of a computer network, for the determination of either performance or status information or to determine the topology of the network.

BACKGROUND

Increasingly, business enterprises require large, complex distributed networks to satisfy their communications and data processing requirements, and many are moving towards implementing large scale computing systems that integrate all the disparate components of the enterprise's computing resources. The efficiency and reliability of communications within these networks is becoming increasingly more important to the overall efficiency of the computing resources. Network management facilities are provided to rectify communications problems, and also to recognize potential problems before they result in communications outages, unacceptable response times, or other impairments (i.e. problem recognition as well as problem resolution). Complex networks often require computer-based systems and network tools to monitor network equipment and facilities, as part of the provision of network management facilities. Concerns about communications' performance and operating costs, and the effects on these variables of node and link failures and reductions in availability, have increased with device and network complexity and sophistication. Hence, the need for monitoring has increased together with the need to enable network reconfiguration from a central location and the generation of alarms when predefined conditions occur.

A highly desirable attribute of network monitoring systems is that they provide the facilities to obtain information, from a single node in the network, about: the state (operational or failed) of any accessible link in the network; the performance of any such operational link (the time taken for inter-node transmissions to traverse that link); and possibly also a specified set of status parameters for each node in the network (in this context, a network node may be either a computer within a network or an application program entity running on the computer).

A monitoring facility is provided in TCP/IP (Transmission Control Protocol/Internet Protocol suite of communications protocols), in the Internet Control Message Protocol (ICMP). ICMP provides error reporting, handling several types of error conditions and always reporting errors back to the original source of the message which led to the error being detected. Any computer using IP accepts ICMP error messages and will change behavior in response to reported errors. Communications links between specific nodes of the network are tested by a first network node (A) sending a timestamped "ICMP Echo Request" message to a second specified node (B). The receiving node (B) then generates a timestamped "ICMP Echo Reply" reply message (reversing the request datagram's source and destination addresses) and transmits it to node A. On receipt of the reply, the node A timestamps the received reply. The time taken to traverse the links (the performance of communication links) between the nodes in each direction can then be calculated. This test facility, known as "pinging" between the nodes, is limited to testing end-to-end performance (from node A to target node B, and vice versa).

U.S. Pat. 5,095,444 describes a system for measuring application message transmission delays in a communications network, providing measurement of delays in the transmission on the various inter-node links of a predetermined communications route between a source node and a destination node. The source node requests (across the communications route) a response from the destination node and a monitor program determines the issue time of the request. The source node then receives the response and the monitor program determines the time of receipt. A transmission delay between the source node and the destination node is determined by calculating a difference between the issue time and the response time. An intermediate transmission delay between any two adjacent intermediate nodes or between an intermediate node and an adjacent destination node is determined by calculating a difference between the transmission delay between the source node and one of the adjacent nodes and the transmission delay between the source node and the other of the adjacent nodes. The source node is required to specify the route between it and the destination node. The system described does not make provision for a changing topology.

EP-A-0510822 describes a system for monitoring node and link status in a distributed network, in which the network monitoring function is distributed among each of the nodes of the network. A first node designated as a dispatching node, dispatches a status table to another node which is on-line. Selected status information about the receiving node is written into the circulating status table (CST) and selected information about the other nodes is read, and the CST is then forwarded to another on-line node. The CST thus circulates around the network according to an adaptive routing sequence, forming a master record of status information which both accumulates and disseminates information about the various on-line nodes of the network. When it has circulated to each on-line node, the CST returns to the dispatching node.

SUMMARY OF INVENTION

The present invention provides a method and a system for monitoring the performance and status of links and/or nodes of a communications network from a single point of control (POC) node, by propagating a test message between the nodes of the network. The method comprises the following steps:

a first process injects into the network a test message requiring specific information by sending the test message to a node test program entity (NTP) on one of the network nodes, the test message including a designation of the POC node for the test;

automatically in response to receipt of the test message, the receiving NTP sends to the POC a reply message including information from the receiving node, and forwards a test message to an NTP on each of said receiving node's own adjacent connected nodes;

each subsequent receiving NTP, automatically in response to receipt of the forwarded test message, also sends to the POC a reply message including information from said subsequent receiving NTP's node, and forwards a test message to an NTP on each of its own adjacent connected nodes.

When the point of control node has received the replies, it can perform an analysis of the received information, compute the performance of any live link (or all live links) or determine the current topology of the network, and possibly display the results and take control actions to modify the network.

In a second aspect, the present invention provides a system for monitoring the performance and status of links

and/or nodes of a communications network from a first Point Of Control (POC) node, by propagating a test message between the nodes of the network, the system comprising:

a process for initiating a test by sending to a first Node Test Program entity (NTP) on one of the network nodes a test message requiring specific information, the test message including a designation of the POC node for the test;

a NTP at the POC node and at every other node of the network, each of which nodes can be a current node for test activity, each NTP including means for receiving the test message and means for performing the following two operations, automatically in response to the received test message: sending to the POC a reply message including information from the current node; and forwarding a test message to an NTP on each of the current node's adjacent nodes;

wherein the POC node has means for receiving the reply messages.

The reply messages received by the POC can then be analyzed to determine the performance of links and nodes and their status (perhaps determining whether links are operative or inoperative and analyzing and displaying results of whatever other node status information was included in the reply messages).

A major advantage of the invention is that a node of the network is not required to know the topology of the network, other than to know its nearest neighbor adjacent nodes, in order to take part in the test. Knowledge of how to address adjacent nodes is a general requirement for message communication rather than a requirement which is specific to the monitoring mechanism of the invention. The test message is similarly not required by the invention to include lists of all the links to be tested, or routing advice for the test. This not only reduces the amount of information which must be included in (i.e. the size of) the test messages but also means that test messages need not be modified if the network is reconfigured. In comparison, a system which relies on a centralized database to provide status and performance information has the considerable overhead of maintaining that database e.g. updating to respond to dynamic changes to the network topology.

Thus, the test message which is initially created preferably does not include propagation routing information, except that it includes some designation of the node (or a program thereon) to which it is initially to be sent. This may comprise a queue name or the network address of the node. Each node preferably has address tables of its local nearest neighbors, which tables are available for the onward transmission of test messages and of other messages. It is, however, preferred that each node knows not only how to reach its nearest neighbors but also how to transmit reply messages to the point of control, both of these information items being part of the setup information specified when the node is configured as part of the network. That knowledge of how to reply to the POC is not essential will be explained later.

It is preferred that each one of the injected test messages, each forwarded test message and each reply message is timestamped when sent by the sender, and that each received message is timestamped on receipt by the receiver. These timestamps, which designate the beginning and the end times of the traversal of each link, are returned to the point of control as part of the reply message so that an analysis process associated with the point of control node can calculate the time taken to traverse each link—i.e. the performance of that link.

Another major advantage of the invention is that individual links that are far removed from the point of control may be tested by the injection of a single message: all nodes reply to the point of control so the point of control accumulates information of the whole connected network. The initially injected message will preferably be sent by a POC process on the POC node to an NTP on either the POC node or on one of its adjacent neighbors. The technique of the invention enables simultaneous monitoring of multiple connections between two nodes (automatically detecting all alternate paths) where multiple connections exist, and enables multiple tests initiated at different nodes to be running simultaneously. The running of a test does not prevent normal communication proceeding simultaneously but merely produces a message flow which adds to the normal network traffic—the monitoring method using specialized test messages which are transferred between nodes in the same way as the normal data transmission messages.

In a preferred implementation of the present invention in a network using message queuing communication between programs (as described later), specific test message queues are used to separate network monitoring test messages from other network traffic to prevent resource contention. In that embodiment, a test message is not addressed to a NTP directly, but to a message queue which is serviced by the NTP. Generally, the test messages are sent to a network node on which some mechanism is provided for ensuring that test messages are received by the NTP.

According to the present invention, the test activity propagates throughout the network in a self-exploring manner, exploiting the parallelism of the network, potentially until all live nodes (i.e. nodes which are not failed) which have live direct or indirect communication links to the point of control have been visited by messages generated within the test and have forwarded reply messages to the point of control. Thus, a single action of injecting a test message into the network results in network-wide information being accumulated at the point of control, unless parts of the network are inaccessible due to failed links or nodes. This is a distinction over certain prior art systems which monitor only the existing network traffic rather than using specialized test messages, as such prior art systems are unable to test links and nodes which are not addressed by normal data transmission messages during the monitoring period. The reply messages according to a preferred embodiment of the present invention, as well as carrying performance information, can equally carry any other status information requested by the point of control when it instigated the test.

The desirability of testing the whole network from a single POC node may depend on the size of the network—if the network is very large and complex then the additional network traffic generated by a test of the whole network will be undesirable unless information of the whole network is actually needed. Thus, it is preferred to define a specific domain within the network which is to be tested from the POC when limitation of the test is desired. Alternatively, a test may be limited by specifying a maximum number of node hops from the POC beyond which the test is not propagated.

Preferably each node of the network has installed thereon at network configuration (e.g. when that node is added to the network) a computer program for monitoring, which is similar for all nodes. This program includes a NTP component and a POC component. Then any node may act as a point of control for monitoring the network.

BRIEF DESCRIPTION OF DRAWINGS

Embodiments of the present invention will now be described in more detail, by way of example, with reference to the accompanying drawings in which:

FIG. 1 is a schematic representation of message queuing communication between processes in a simple distributed computer network;

FIG. 2 is a representation of the data fields of an example data transmission message;

FIG. 3 is a flow diagram showing the steps taken by a Point Of Control process in the execution of a network monitoring method according to an embodiment of the present invention; and

FIG. 4 is a flow diagram showing the steps taken by a Network Test Program on receipt of a network monitoring test message, according to an embodiment of the network monitoring method of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The problem of how to obtain performance and status information about the nodes and interconnecting links of a computer network from a single node of the network occurs in messaging networks—where participating nodes communicate by the asynchronous passing of messages between adjacent nodes in the network. The facility in such systems for inter-process data exchange is often based on management of message queues. The message model is simple: a sending process enqueues messages by putting to a queue (the sending process issues a "Put Message"-type command); and then a receiving process dequeues (issuing a "Get Message"-type command to take the message from the queue either for processing or transferring onwards towards the destination). The enqueue and dequeue operations are performed asynchronously, dequeue being when the receiving process chooses rather than at a time dictated by the sender. Queue managers deliver each message to the proper queue associated with the destination process; it is the network of interconnected queue managers that is responsible for moving messages to the intended queues.

Message queuing is thus a method of inter-program communication which allows programs to send and receive application-specific data without having a direct connection between them, and without necessarily being operational simultaneously. Application programs can run independently of each other, at different speeds and times. The application programs communicate by agreeing to use particular named message queues, sending messages to the specific queues that the target programs have agreed to read from. The locations of the queues are not generally apparent to the applications which send the messages; each application interacts only with its local queue manager. Applications are thereby shielded from network complexities and variations by the queue managers. All of the work involved in maintaining message queues, in maintaining the relationships between messages and queues, in handling network failures and restarts, and in moving messages around the network can be handled by the queue managers. Since cross-network communication sessions are established between queue managers rather than between individual programs, programs are less vulnerable to network failures than in other types of communication.

A message queue is thus a named object in which messages accumulate and from which they are later removed, which is maintained by a particular queue manager. The physical representation of a queue depends on the environment (e.g. it may be a buffer in main storage or a file on disk). A message queue is a storage means whereby messages are normally added and removed in FIFO order, but facilities also exist allowing messages to be read in other than the order in which they arrive on the queue.

Such message queuing communication is further described in IBM Document Number GC33-0805-00 "IBM Messaging and Queuing Series: An Introduction to Messaging and Queuing", and is implemented in the IBM Messaging and Queuing Series products (see the "IBM Messaging and Queuing Series Technical Reference", IBM Document Number SC33-0850-01) such as the IBM Message Queue Manager MVS/ESA.

A schematic representation of message queuing communication is shown in FIG. 1, within a computing network which comprises a set of nodes 10 and interconnecting links 20. There may be a plurality of different physical communication paths between nodes of the network. In this context the "nodes" are the computers of a network, but unless otherwise required by the context the word "node" is intended in the following description to apply equally to program entities running on those computers. The present invention can monitor the status and performance of interconnected computers or of individual communicating processes (e.g. application programs) within the same computer system or between different systems (in the latter case providing an application-connectivity test tool).

A first process A (30) transmits messages to other processes B,C,D by putting messages to message queues (40, 50,60) which the target processes have agreed to receive messages from. The process A simply issues a Put Message command, specifying the name of the destination queue (and possibly the name of a particular queue manager which is responsible for managing that queue). A message delivery system (queue manager) 70 on the computer system on which process A is located is responsible for determining where to route incoming messages and where the messages from A should be sent for example to the incoming queue 40 of another process B on the same computer or across the network to a queue (50 or 60) of a remote process (C or D). In actuality, a message sent to a remote node is transferred in multiple hops between queues on adjacent nodes enroute to the destination node, but this is invisible to the origin and destination processes. Each remote process (C or D) uses a message delivery system (queue manager) which is responsible for putting all incoming messages to the appropriate queue for that process. A queue manager may be responsible for many queues. The processes C and D take the messages from their input queues 50 and 60 when they are ready.

The present invention is particularly applicable to networks using message queuing communication, enabling the monitoring of message queue parameters such as the number of messages on (i.e. the size of) queues, the time taken to service a queue, and other data of the throughflow of messages. Such information is very valuable for systems management, for example for enabling the determination of whether load balancing network reconfiguration is required (e.g. increasing or decreasing the number of application program instances servicing a particular queue).

One aspect of the invention which is particularly relevant to network management in a message queuing system, is the ability of the monitoring method of the present invention to provide performance information for links between nodes in both directions —which may be different from each other since the throughflow of messages across a link depends on

the nature and number of application instances servicing (receiving messages from) the queue and the number of applications which are putting messages to the queue, rather than just the limitations of the communications protocol and the capacity of the underlying hardware links.

As is known in the prior art, a message consists of two components —an application data component **100** and a data structure called the message header **110**, containing control information, as shown in FIG. 2. The application data in a message is defined and supplied by the application which sends the message. There are no constraints on the nature of the data in the message (for example, it could consist of one or more bit strings, character strings, or binary integers: if the message is a data transmission message relating to a financial transaction, the application data items within the message may include, for example, a four byte unsigned binary integer containing an account number and a twenty byte character string containing a customer name).

In addition to the application data, a message has associated with it some ancillary data. This is information that specifies the properties of the message, and is used on receipt of the message to decide how the message should be processed. The ancillary control information is contained in the message header **110**. Some of this information must be specified by the sending application program. With this message structure, routing information may be included in the message header so that it is possible to determine from the message header whether the message is destined for the local node (e.g. for an application program running on a computer which receives the message) or for an adjacent or remote node of the network, whereby to route the message. In principle, there is not a specific distinction between the information which can be contained in the data portion and that which can be contained in the header portion— information of the origin, destination, time of sending, etc., may be in either portion.

Where the routing information is included in the message's header portion, messages are generally transferred between intermediate nodes enroute to a destination node without the application data being analyzed (or possibly even reformatted) by the intermediate nodes. If different operating systems are running on the various computers of the network, then communication between computers requires, however, reconfiguration of messages according to conventions of a network protocol stack existing between them.

According to the present invention, a test message is a specific type of message which is distinguished from application data messages in that it includes in its message header either: the name or other designation of a queue which is specific to test messages on the target node, so that the receiving queue manager can route the test message appropriately; or a flag by which the receiving node can identify it as a test message without reading the message's data content. The former is the preferred implementation. Three different types of "test messages" may be identified which are associated with the monitoring function of the present invention. These are described below.

Each node of the network has a network management program, which can be identical for each node and which provides the network monitoring facility, installed thereon. The network management program provides the means for propagating a test message throughout the network from a single point of control, in a self-exploring manner, so that the test reaches all of the connected nodes of the network and each node replies to the point of control.

Each node of the network has means for maintaining a list of its adjacent nodes, in a table which contains the names and addresses of the computers (and possibly also the processes thereon or their associated queue names) which have direct connections to the node in question. For a given node the local network manager program has access to the list of adjacent nodes, as a source of information on how to communicate with these adjacent nodes. This list is generated when the system is configured, or more precisely when the particular node is configured as part of the system, and is updated when a new adjacent node is added to the system. Reconfiguration can be done dynamically without taking the relevant nodes out of service. On addition of a new node to the network, that node may broadcast its configuration information to its connected nodes.

A simple naming convention is adopted in that the input queue for node's system management program is: Node_Name. Test_Queue. The test program of the invention then puts messages to "Node_Name. Test_Queue" for each of its neighbor nodes in order to send messages to them. This is sufficient to get to the desired destination neighbors since they are known to the current node.

The network manager program also has the ability to store a global test identifier of the tests for which it receives messages. The network manager program thereby forms a record of test identifiers that can be referred to on receipt of a further test message to determine whether the node has already participated in the particular test of which that new message is a part. The importance of the global test identifier is explained below. Because all nodes have installed the same (or an equivalent) network management program, any node can be the point of control for a particular test instance i.e the node to which test information is returned, and in the preferred embodiment of the invention also the node from which the test is initiated.

The logical steps of the network monitoring function of the network management program for each node are represented in FIGS. 3 and 4. The network management program has two effectively independent components or subroutines which provide the monitoring facility and which can send messages to each other: the first component, referred to as the POC process, provides the means for the node to operate as a point of control (POC) node, initiating a test and receiving responses to the test; and the second, referred to as the network test program (NTP), provides the means at each node to process an incoming test message and then to perform the operations required by the test message. Each node's POC and NTP has its own incoming message queue or there is a single incoming queue for both the POC and NTP. FIG. 3 represents the process steps of the POC and FIG. 4 represents the process steps of the NTP.

In practice, it will generally not be necessary simultaneously to have a plurality of nodes each acting as the single point of control for a different test. However, the facility for multiple POCs is provided by the present invention and may be desirable for enabling different nodes to monitor different network performance parameters. It is much more likely that a network administrator will require only a single POC at any one time to which all test responses are sent, but that the administrator will wish to change which node is the POC on occasion (for example when the current POC is to be disconnected from the network). This is the reason for providing each node with both the POC and NTP components of the network management program.

Monitoring according to the present invention will be of maximum benefit if performed at regular intervals. Thus, a

test may be initiated by a timer, for example a new test beginning automatically at the end of the previous test period, but it may equally be initiated in response to a specific request for information from one of the network nodes, or it may be triggered in any other way. When the POC process is triggered **200** to initiate a test, the POC process at the point of control node sends **210** a specific test message to a single first NTP, which is preferably at the POC node but may be at one of its adjacent neighbor nodes. This first message is timestamped as sent i.e. an entry is made in a specific data field. The message also includes a reply address (i.e. the name of the single POC) for the return of reply messages, and a global test identifier **204**. The global test identifier is a simple alphanumeric string comprising the node name of the POC, the time of day at which the initial test message was created, and an incremental sequence number. The global test identifier may alternatively be any identifier which is unique for the particular test during the test period and for the network or part of the network which is specified to be tested by the test.

The POC node also specifies **206** an expiry time for the test, which expiry time T is included in the test message sent to the first NTP. This inclusion is advantageous, as will become clear later, because a failed node which receives a test message in its input queue and then becomes operational at some later time can simply delete a message which is clearly part of an expired test. An alternative to this which requires only the POC to know the expiry time of the test is to dispose of messages only at the POC, so that out-of-time messages are returned to the POC. The expiry time is generally not a critical parameter—its selection is determined by the time from the initiation of the test after which the replies to the POC are no longer useful. It is likely that after a certain time any further replies from the network will be considered unacceptably slow, and in such circumstances it will be necessary to take some corrective action but it may be unnecessary to quantify degrees of unacceptability.

The POC's initial message may include further information, depending on what information is required by the particular test instance. The test message itself includes specific data fields for the network administrator to specify what information is required to be included in reply messages. Thus, the administrator can customize or vary what is being monitored. Examples of what information may be required have already been given. In an alternative embodiment of the invention, the test message itself may have no such optional data fields and so does not specify what information is required by the test the NTP programs are instead provided with instructions on set-up of what information is to be included in all reply messages. This alternative is only acceptable if the type of test information required by the POC will always be the same. In implementing this alternative, the node which receives a test message simply needs to identify that a test message has been received and the name of the POC to enable it to construct a standard-structure reply message for sending to the point of control (see later with reference to FIG. 4).

Having sent the first message to the first NTP with instructions on how to reply to the POC, the POC waits **220** for the specified time T. On expiry of time T, the POC inspects **230** all reply messages which are part of the specific test (visible from the global test identifier which is part of all messages within the test) and performs an analysis **240**. The POC can then produce tables of performance and status information for display to the network administrator.

The operation of the NTPs will now be described with reference to FIG. 4. On receipt of a test message, each NTP performs the same operations, whether it is the NTP in the POC node receiving the first message sent from the POC or whether it is an NTP on a remote node receiving a test message which has been propagated across many nodes. When a node receives any message from one of its adjacent nodes, a receiving process owned by the queue manager examines the message header to determine where to route the message. If the message is a data message destined for an application program running on the local node then the receiving process directs the message to the incoming message queue for the local application, for retrieval when the application is ready. If the destination application program is a remote application then the receiving process directs the message onwards towards its destination using destination address information in the message header. If the message is a test message, then its header information will specify the receiving node's NTP input queue as its destination and the message will be routed accordingly. In an alternative embodiment which is not a message queuing implementation of the invention, test messages each include a flag which identifies them as a test message. This flag triggers a receiving process on the node to call the local NTP to process the test message.

When a test message arrives on a NTP's incoming message queue, the local NTP will be triggered **300** to retrieve and process it. The NTP first records **310** the time of receipt of the Received Test Message by the current node. It is then determined **315** whether the Received Test Message is part of an expired test, by checking whether the assigned expiry time T has passed. If the test has expired, the local NTP exits (after first deleting from its list of "known" global test identifiers any entry it may have for the relevant global test identifier). If the test has not expired, then the NTP creates **320** a Reply Test Message to return to the POC. As noted earlier, either the received message includes details of what information the Reply Test Message should include or the NTP is given instructions on this during set-up. The queue managers each include means for running processes for measuring performance and status parameters such as message throughput, and the NTP uses an API call to obtain this information. The API can be used to set up limits for triggering specific responses to the POC: for example parameters such as queue depth may have limits set, with a notification being sent to the POC within a Reply Test Message when any received messages extend the queue depth beyond the limit.

The time of receipt of the Received Test Message is included in a data field of the Reply Test Message (i.e. the current node timestamps the test messages before transmitting them). If the test is of link performance only (i.e. no other information is required than times to communicate messages between network nodes), the Reply Test Message may be little more than a notification of receipt bearing the global test identifier and the name of the POC, the node names of the previous and current node and the times of sending and receipt of the Received Test Message. The Reply Test Message thus need only include the information of the Received Test Message plus the timestamp of the time that the message was received at the current node, and the name of the current node. However, the Received Test Message preferably has a plurality of data fields which may be used when the test is initiated to specify information that is required by the test, and then that information will be included in Reply Test Messages. The network management program is designed to perform timestamping automatically on receipt of a test message at the local node, without regard to specific data fields of the test message.

The Reply Test Message will generally include information from the current node about how to reach the POC, which information is provided at the current node during its configuration. Alternatively, each NTP may keep a record of all received messages including information on which of its adjacent nodes sent each message (for example, the name of the node which sent the message may be recorded together with the global test identifier). In that case it is possible to relax the requirement of having to know the POC location—a Reply Test Message can simply be sent to the previous adjacent node (from which the current Received Test Message was sent) which will then send it on towards the POC, retracing the hops of the earlier propagation. Since it is true for all nodes that participate in the test that there is a path to the POC via the previous nodes involved in the propagation, the Reply Test Message can be passed back along that path. (Of course, there is a bi-directionality assumption here that, if a link from A to B is operational, then the link from S to A is also operational within the period of the test. This assumption is not required if the POC location is known to the participating nodes.)

It is then determined **330** for Received Test Messages whether the global test identifier is known to the node. Each node has a list of the global test identifiers of tests in which the current node has already participated, and this list is examined to determine whether the local node has previously received and replied to a test message associated with the current test. If the current node has previously participated in the test in this way, the local Network Test Program instance is exited **340**. The global test identifier thus serves to ensure that each node only propagates the test (see below) once for a given test. This is important because, for networks of complex topology, it is probable that a given node will receive a number of Propagated Test Messages within an individual test as all the links to that node are tested; exiting the program after the local test is honored, avoids unnecessary duplication of message flow.

If the test of the Received Test Message is new to the current node, then the global test indicator for the node is recorded **350** in the node's list of global test identifiers. Then a timestamped Propagation Test Message is created **360** at the current node. The Propagated Test Message has the same format as the Received Test Message, except that in relation to the current node the Propagated Test Message includes the name of and time of sending by the current node whereas the Received Test Message includes the name of and time of sending by the previous node. Each Propagated Test Message has the same structure (i.e. number of data fields) as the first—there is no need for the propagated test messages to include accumulated information of all of the network nodes which have participated in the test, and instead information for the current node replaces in the Propagated Test Message the information of the previous node that was included in the Received Test Message.

The Propagated Test Message is then sent **370** to all nodes adjacent the current node and becomes a Received Test Message at each of their NTPs. The test is thereby propagated to the rest of the network. This propagation technique exploits the parallelism of the network as a large number of nodes may be simultaneously working on the same test. A given queue manager in administering message flow is not required to know how to get to a queue or queue manager (except possibly on the POC) that is more than one node "hop" away. The node test program at a given node simply puts messages to the named test message queue ("Node_ Name. Test_Queue") for each of its immediate neighbor nodes. Having sent the reply, stored the global test identifier and propagated messages to its adjacent nodes, the local instance of the Network Test Program is then ended **380**.

It is possible for a given node's NTP and POC processes to receive messages from the same incoming queue, although this is not preferred. In that case there must be a mechanism for distinguishing between Reply TestMessages and Propagated Test Messages, and for determining whether the current node is the point of control node for the test. If the message is a Reply Test Message destined for the current node then its information content is stored for analysis. If the test message is not a Reply Test Message then, in response to its receipt, new messages are created by the receiving node for forwarding to adjacent nodes and a Reply Test Message is created for sending to the POC.

As already discussed above in terms of the operations performed by the POC and the various NTPs for a given test, there are three types of test messages involved in the network monitoring method. These three types of test messages are as follows:

1) The Propagated Test Message sent by a node to an adjacent node, which contains:

the global test identifier

the name of the single point of control node

the name of the node which sent the Propagated Test Message

the time when the Propagated Test Message was sent.

The initial message sent from the POC has the structure of a propagated test message, but is rather different in that it is only sent to one node instead of all nodes adjacent to the sending node and in that it is created entirely by its sending (POC) node rather than being generated by processing a Received Test Message.

2) A Received Test Message received at the current node from an adjacent node, which contains:

the global test identifier

the name of the single point of control node

the name of the previous node that sent the Received Test Message

the time when the previous node sent the Received Test Message.

A Propagated Test Message and a Received Test Message are the same entity, and the distinction is in whether their information content relates to the current node or to the previous node.

3) Reply Test Messages, which are sent to the point of control from any node receiving a test message within the current test, which contain:

the global test identifier

the name of the single point of control node

the name of the previous node that sent the Received Test Message

the time when the previous node sent the Received Test Message

the time the Received Test Message was received at the current node

the name of the current (receiving) node.

The global test identifier, the name of the single point of control for the test, the names of the previous sender node and of the current node, and the time information are preferably included in the message header, but may alternatively be in the data part. In addition, the Reply Test Messages can carry status information from the node in question, which is part of the message's application data component. This status information component is preferably

subject to filters and is only included on the first reply from the node, for a given global test. Examples of the status information that the test may require are queue sizes for each node at a point in time, or throughput of messages over time, or the period of time which a message remains on a particular queue or on a particular node.

It should be noted that a reply of sorts is sent to the previous node as well as the Reply Test Message sent to the POC, in the form of one of the Propagated Test Messages. Thus a POC node will receive a Reply Test Message and a Propagated Test Message from each of its adjacent nodes.

Since the individual nodes are not aware of the network topology, it is not very practical to seek to determine when all nodes have processed a Received Test Message for the particular test and when all replies have been received by the POC, nor is it necessary. Instead the propagation and reply phase of the test is considered to be complete when the set time period T expires. The POC then dequeues from its incoming message queue all Reply Test Messages having the global test identifier of the completed test. Of course, the POC may alternatively dequeue Reply Test Messages as they arrive and process the messages at that time to create a table of the received information. The POC process, or another process available for use by the POC, can now move into an analysis phase, where it computes performance or other statistics or determines the topology of the logical network from the point of control. All connected nodes of the network participate in the test and reply to the point of control. By subtraction, the point of control node's analysis process can determine the performance of each node of the network in both directions.

The performance information displayed to the network administrator will generally be actual times to traverse network links or an indication that the time was greater than the time limit T. Thus node or link failures are presented as performance information rather than configuration information, which is more useful for network administration. No distinction is made in the information accumulation stage of the test between node and link failures—each is noted by the time to reply being greater than T. However, analysis of the test output can provide such information in most cases (e.g. if only one of a number of links to a node fails then that this is not a node failure will be clear from the other responses received at the POC).

It should be noted that the method of monitoring according to the present invention has some reliance on the synchronization of the clocks in the participating nodes. However, the computation of times taken to traverse a link in each direction eliminates or reduces this problem, as long as the times that the test messages remain on queues between the NTPs is not significant. This is a reasonable assumption for the preferred embodiment of the present invention which employs specialized queues for the test messages so that the test messages do not await the dequeuing of simultaneously transmitted application data messages. Any distributed network needs some timer services and all nodes will generally have clocks. Provided that the clocks are synchronized to within a period of time which is much shorter than the test expiry time T, the present invention is not compromised. It is reasonable to assume that some method of synchronizing the clocks is provided.

Thus, in the preferred embodiment of the invention, the timestamping of test messages enables direct measurement of link performance between adjacent NTPs, but the performance information for application data messages (taking account of the length of time which they remain on message queues between processes) is obtained indirectly via the API—the NTP accessing the measurements made by the various queue managers and including this information in the Reply Test Messages.

In the alternative that there are not specialized queues for test messages, the timestamped test messages directly obtain information of the performance of application message queues as they utilize those queues. In such an embodiment, the synchronization problem can be solved by periodically running tests for which the test messages are given a higher priority than application messages such that the tests are retrieved from the queues in advance of the application messages (i.e. testing as if the queues were empty of application messages).

Testing according to the present invention may be performed at regular intervals, thereby producing trend information that will have significant value in managing the network. If the required analysis of the information accumulated at the point of control is to compute trends rather than nominal performance values, then there is no reliance on the synchronizing of clocks in the participating nodes despite the time distortions introduced by the synchronizing problem. Alternatively to regular periodic testing, a test may be initiated when a node is notified of some reconfiguration of the network or when a user requests information.

In another alternative implementation of the present invention, each node is only required to send a Reply Test Message to the POC on the first occasion that a message having a particular global test identifier is received. In this way test message traffic is substantially reduced. This embodiment of the invention is satisfactory, for example, where a test is not required to directly measure performance but instead obtains all the information for its Reply Test Messages from the node's queue manager via its API or from some other process.

A consequence of the technique of the present invention using existing network links to test the network should be noted: if link or system failures partition the network into isolated pieces, then only limited information may be received by the single point of control—only nodes that share the same partition as the point of control will respond to the test. However, the failure can be identified to the point of control to enable resolution of the problem.

Partly to reduce the problem mentioned in the last paragraph and additionally because it is rare in the management of large networks for information of the whole network to be required at a single node, a network may be split into specific domains for testing. That is, domains are defined and each node and link is either within or outside the particular test domain. If a node outside the test domain receives a test message, no propagation or reply message will be created. For a large network, this can substantially reduce unwanted network test traffic and in particular reduce the number of expired test messages in the system. An alternative to domain definition, which does require some knowledge of the network topology, is to set a limit for the maximum number of hops from the POC which are to be tested in a given test. A network administrator may also wish to conduct higher level testing on occasion, such as a test of all of the different domains POCs. This is possible if the POCs are enabled to identify themselves when a propagated message of such a high level test is received. A test may alternatively request responses only from specific named nodes.

Having thus described my invention, what I claim and desire to secure by Letters Patent is as follows:

1. A system for monitoring the performance and status of links and/or nodes of a communications network from a first Point of Control (POC) node, by propagating a test message between the nodes of the network, the system comprising:

means for initiating a test by sending to a first Node Test Program (NTP) on one of the network nodes the test message requiring specific information, said test message identifying the POC node for the test;

a NTP at the POC node and at every other node of the network, wherein each node can be a current node for test activity, each NTP including means for receiving the test message and means for performing the operations of sending to the POC node a reply message including information from the current node, and forwarding the test message to the NTP node on each of the current node's adjacent nodes, said operations being done automatically in response to the received test message;

means for receiving said reply messages at the POC node; and means associated with said POC node for analysis of said rely messages.

2. The system according to claim 1 wherein the means for initiating a test includes means for setting an expiration time for the test, which expiration time is included in all forwarded messages and is recognizable by the receiving NTP to indicate that the message should be deleted if the test has expired, without creation of messages by the NTP for forwarding to adjacent nodes or for reply to the POC node.

3. The system according to claim 1, wherein each node of the network is provided with a POC process to perform the operation of the POC node of receiving reply messages to the test.

4. The system according to claim 1 or claim 3, wherein the POC node has program means for initiating tests by sending the test message to the NTP at the POC node or at an adjacent node.

5. The system according to claim 1 or claim 3, wherein the NTP at each node of the network, as the current node for test activity, includes means for timestamping messages on sending and on receipt by the current node, and for including in said reply messages sent to the POC node information of the time the test message was sent by the previous node and of the time the test message was received by the current node.

6. The system according to claim 5, including means associated with the point of control node for analyzing the time information of the reply messages to compute the performance of particular nodes and links of the network.

7. The system according to claim 1 or claim 3, including means associated with the point of control node for analyzing the information of the reply messages to determine the topology of the network from the point of control node.

8. The system according to claim 5, including means for exiting the current node's NTP on receipt of the test message which is part of the test for which the current node has previously received the test message, without forwarding test messages to NTPs on the adjacent nodes.

9. The system according to claim 8, wherein the POC node program means for initiating a test includes means for setting a global test identifier for the test message, which test identifier is unique to the particular test of the current message and is identifiable by NTPs on receipt of the message to identify the particular test, wherein each node of the network has means for maintaining a list of the global test identifiers for tests which are known to the node.

10. A method for monitoring the performance and status of links and/or nodes of a communications network from a single point of control (POC) node, by propagating a test message between the nodes of the network, the method comprising the following steps:

injecting into the network the test message requiring specific information by sending the test message to a node test program (NTP) on one of the network nodes, said test message identifying the POC node for the test;

automatically, in response to receipt of the test message, sending to the POC node from the receiving NTP a reply message including information from the receiving node, and forwarding the test message to the NTP on each of said receiving node's adjacent connected nodes;

each subsequent receiving NTP, automatically in response to receipt of the forwarded test message, at least if the received test message is within an unexpired test, sending to the POC node a reply message including information from said subsequent receiving NTP's node, and, at least the first time said subsequent NTP receives a message within a given test, forwarding the test message to the NTP on each of its own adjacent connected nodes; and

analyzing reply messages received by the POC node.

11. The method according to claim 10 including the step of setting an expiration time for the test when the test is initiated, which expiration time is then included in all forwarded messages and is recognizable by the receiving NTP to indicate that the message should be deleted if the test has expired, without creation of messages by the NTP for forwarding to adjacent nodes or for reply to the POC node.

12. The method according to claim 10, wherein the test message is injected into the network by a first POC process on the POC node which sends the test message to the NTP on the POC node or on one of the POC node's adjacent nodes.

13. The method according to claim 10 or claim 12, wherein the test message is timestamped on sending and on receipt by a current node, and information of the times that the test message was sent by a previous node and received by the current node are included in reply messages sent to the POC node.

14. The method according to claim 13, wherein the time information of the reply messages is analyzed by a process associated with the POC node to compute the performance of particular nodes and links of the network.

15. The method according to claim 10 or claim 12, wherein the information of the reply messages is analyzed by a process associated with the point of control node to determine the topology of the network from the point of control node.

16. The method according to claim 10 or claim 12 wherein the NTP only propagates the test message to the receiving node's adjacent nodes on the first occasion that the node receives a propagated test message within the particular test.

17. The method according to claim 16, wherein the POC process sets a global test identifier for the test message when it initiates the test, which test identifier is unique to the particular test of the current test message and is identifiable by NTPs on receipt of the test message to identify the particular test, each node of the network maintaining a list of the global test identifiers of tests for which messages have been received by the current node.

18. A computer having means for monitoring the performance and status of links and/or nodes of a communications

network which includes the computer, the computer having a network management program thereon which comprises:

a Point of Control (POC) means for initiating a test by sending to a first Node Test Program (NTP) on said computer or on a connected computer a test message requiring specific information, said test message identifying the computer as the POC node for the test, the POC process including means for receiving, from NTPs on the computer and on other computers with direct or indirect connections thereto, reply messages to a test for which the computer is the identified POC node; and

a Network Test Program (NTP) for processing a received test message automatically on receipt by performing the following operations: sending to the POC node identified by the received test message a reply including information from the current computer; and forwarding the test message to the NTP on each of the adjacent computers with direct connections to the current computer;

said network management program further comprising means associated with the POC means for analyzing said reply messages.

* * * * *